Louisiana State University
LSU Digital Commons

LSU Doctoral Dissertations | Graduate School

2003

Control of transport dynamics in overlay networks

Qishi Wu
Louisiana State University and Agricultural and Mechanical College, qwu2@lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

 Part of the Computer Sciences Commons

Recommended Citation
Wu, Qishi, "Control of transport dynamics in overlay networks" (2003). *LSU Doctoral Dissertations*. 3783.
https://digitalcommons.lsu.edu/gradschool_dissertations/3783

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contactgradetd@lsu.edu.

CONTROL OF TRANSPORT DYNAMICS IN OVERLAY NETWORKS

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
The Department of Computer Science

by
Qishi Wu
B.S., Zhejiang University, P.R. China, 1995
M.S., Purdue University, 2000
May 2003

Copyright 2003
Qishi Wu
All rights reserved

This dissertation is dedicated to my beloved wife, Mengxia.

ACKNOWLEDGEMENTS

On the day after the graduation ceremony at Purdue University in 2000, I drove to Baton Rouge by myself all the way from West Lafayette, Indiana to meet with my girlfriend and start my Ph.D. study at LSU. I am happy that I made the decision to come to LSU because these past few years have been the most challenging, fulfilling, and exiting moment in my life. I have not only achieved success in personal life but also reached my academic expectations: my girlfriend has become my wife and now I am completing my Ph.D. degree. I would like to use this opportunity to express my great gratitude to people who have helped me in the past and present to get where I am now.

I would like to first thank my major advisor, Dr. Iyengar, for bringing me to LSU and guiding me throughout my Ph.D. studies. I have not only learned the beauty of science from him, but the importance of honesty, integrity, and hard work as a researcher by his own example. More importantly, I have created broad professional relationships with prestigious researchers at other universities and institutes thanks to his efforts. Dr. Iyengar has been also very supportive and considerate in many non-academic aspects, which actually have had significant impacts on my life as an international student.

I have spent totally about one year at ORNL working closely with my co-advisor, Dr. Rao, on my dissertation in computer networking. Dr. Rao has been the constant source of inspiration whenever I needed direction in my research. The dissertation work outlined here is not separable from his insightful and nurturing ideas generously shared with me. His genuine erudition, profound expertise, and great personality have set a lifetime example for me. I acknowledge him for all his help with my deepest gratitude.

I am highly honored to have Dr. Aravena from Dept of Electrical and Computer Engineering as the minor professor, Dr. Kraft and Dr. Jones from Dept of Computer Science as the committee members, and Dr. Kurtz from Dept of Physics as the dean's representative in my Ph.D. committee.

Dr. Aravena deserves the particular acknowledgement for his valuable comments and suggestions that greatly improved the technical quality of this dissertation.

I am also indebted to Susanne Levine, and thank her for proofreading part of my dissertation and making many suggestions and corrections that improved the appearance of the final document.

I had the privilege to conduct and complete my dissertation work at the Oak Ridge National Laboratory, which is one of the most elite research facilities in the States. I would like to thank the ORNL for providing me the opportunity to work and the access to research resources in the lab. The research outlined in this dissertation is funded by the Department of Energy High-Performance Networking Program through the ORNL to Professor Iyengar.

Finally, I would like to thank all the spiritual and physical supports from my wife, parents, sister and brother. Their unconditional, unchanging, and unending love and care has been the strongest force for me to be the best that I can be and will accompany me for all my life to achieve much more in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Transport control is an important factor in the performance of Internet protocols, particularly in the next generation network applications involving computational steering, interactive visualization, instrument control, and transfer of large data sets. The widely deployed Transport Control Protocol is inadequate for these tasks due to its performance drawbacks. The purpose of this dissertation is to conduct a rigorous analytical study on the design and performance of transport protocols, and systematically develop a new class of protocols to overcome the limitations of current methods.

Various sources of randomness exist in network performance measurements due to the stochastic nature of network traffic. We propose a new class of transport protocols that explicitly accounts for the randomness based on dynamic stochastic approximation methods. These protocols use congestion window and idle time to dynamically control the source rate to achieve transport objectives. We conduct statistical analyses to determine the main effects of these two control parameters and their interaction effects. The application of stochastic approximation methods enables us to show the analytical stability of the transport protocols and avoid pre-selecting the flow and congestion control parameters. These new protocols are successfully applied to transport control for both goodput stabilization and maximization. The experimental results show the superior performance compared to current methods particularly for Internet applications.

To effectively deploy these protocols over the Internet, we develop an overlay network, which resides at the application level to provide data transmission service using User Datagram Protocol. The overlay network, together with the new protocols based on User Datagram Protocol, provides an effective environment for implementing transport control using application-level modules. We also study problems in overlay networks such as path bandwidth estimation and multiple quickest path computation.

In wireless networks, most packet losses are caused by physical signal losses and do not necessarily indicate network congestion. Furthermore, the physical link connectivity in ad-hoc networks deployed in unstructured areas is unpredictable. We develop the Connectivity-Through-Time protocols that exploit the node movements to deliver data under dynamic connectivity. We integrate this protocol into overlay networks and present experimental results using network to support a team of mobile robots.

CHAPTER 1 INTRODUCTION

1.1 Overview

The early pioneers of computer networks might not have imagined that in less than thirty years the world would have been changed enormously by the groundbreaking ideas they conceived. Among a variety of existing networks, the Internet is the most rapidly growing communication medium, which has virtually turned our planet into a "global village".

Most of the present computer networks are built on similar class of protocols, which are essentially a set of rules created to facilitate communications among various types of computers and electronic devices. These protocols were initially designed with the consideration of hardware limits in mind at an early stage of network development. In recent years, new technologies in computing and communications have been evolving so quickly that the physical medium bandwidth limit is not the main concern any more for most current network applications. However, network performance is still not as good as we expected at the application level because of the limitations of the initial design methodology of these protocols, especially those in the transport layer.

The main purpose of the research outlined in this dissertation is to conduct a rigorous analytical study on the design and performance of transport protocols, and systematically develop a class of acceptable solutions based on statistical analysis methods and the current Internet implementations. These protocols are realized through daemons based Overlay Networks (ON), which reside at the application level and perform Transport Controls Over UDP (TCOU). The proposed Overlay Network with Transport Controls Over UDP (ONTCOU) is expected to achieve analytically validated and experimentally at least as good as the default TCP or UDP in terms of attributes such as throughput, stability, dynamics, and fairness. In light of the fact that wireless networks have quite different link characteristics than wired networks, further research efforts are also made in adapting ONTCOU to wireless network environments to resolve connectivity-related transmission control issues.

1.2 Network Performance Metrics

In this section, we introduce some concepts in network transport control and technological terms used for network performance evaluations, such as delay, RTT, bandwidth, throughput, goodput, packet loss, congestion collapse, which appear in this dissertation.

1.2.1 Delays in Computer Networks

In computer networks, delay, a synonym for latency, denotes the amount of time it takes for a packet of data to get from one designated point to another. In reality, every single bit experiences various types of latencies during its transmission in a network. The main contributors to network latency include the following:

- **Link Propagation Delay**

Link propagation delay simply represents the time it takes for a packet to travel through a wireless or wired link at the finite, constant speed of light, and is solely determined by the

physical link distance between the source and destination nodes. The propagation delay within a Local Area Network (LAN) may not necessarily be considered due to its negligibly short distance compared to the light speed. However, it could become a significant part of latencies in cross-country or intercontinental satellite links. The data packet transfer time over the Internet is ultimately limited by the speed of light, no matter how much technological progress such as protocol improvement and bandwidth enhancement is made.

- **Transmission Equipments Associated Delay**

At each node of the data delivery path including source and destination nodes within a packet-switching network, an incoming packet is stored in some cases temporarily in an input queue until the routing algorithm finds a proper output channel and the switching function establishes the physical connection for it, while an outgoing packet is usually stored temporarily in an output queue until its signaling turn arrives. The amount of time a packet waits in the router queues largely depends on how busy the network is at the moment. Since the network state is always subject to continuous and dynamic changes, the queuing delay is the most complicated and unpredictable type of delay incurred during the data transmission.
In a LAN of a certain type of topology, such as bus topology (Ethernet) or ring topology (IBM token ring, FDDI), there may also be a Media Access Delay (MAD) between two contiguous transmissions because the transmitting station must ensure that it has an exclusive occupancy of the physical communication medium of the LAN. Multiple users contending for the limited hardware resources simultaneously are the main cause for transmission equipments associated delays.

- **Bandwidth-constrained Delay**

All the present communication media, such as twisted pair, coaxial cable, optical fiber, microwave, infrared, laser, have their own bandwidth limitations, which imposes a lower bound on the time in which a packet can be transmitted. Generally speaking, under similar network conditions, a larger packet always takes longer to travel along the same path than a smaller one. This type of delay is determined by the message size and effective bandwidth. Recent technological advancement shows that there is a lot more room for improvement in the bandwidth-constrained delay than in other types of delays. Although the rapid progress of new technologies, particularly in optical network components, has offered plenty of bandwidth at the link level, the efficiency of using the available bandwidth is still the key to achieving good Quality of Service (QoS).
It is universally believed in the computer networking community that the protocols in the transport layer play a significant role in overall network performances such as bandwidth utilization efficiency. This is the reason the design issues of transport layer protocols have been attracting a great deal of attention from many network researchers. The research presented in this dissertation also focuses on the design and performance issues of transport layer protocols that are implemented in overlay networks.

1.2.2 Round-Trip Time

Round-Trip Time (RTT) is the sum of two-way latencies between two end hosts, which consists of the time taken for a single packet to leave one machine, pass by the intermediate routers, reach the other end, and return. Since most of the routing tables in

the Internet remain stable for a few days or weeks, the packets in one transmission session with a normal duration usually take the same route to get to the destination. However, the travel path is very likely not the same in both directions between two hosts because routing in the Internet is often asymmetric [Paxson96]. Figure 1.1 shows the two-way directional routes between two hosts, resource.rrl.lsu.edu located at LSU and ozy4.csm.ornl.gov located at ORNL, which was provided by traceroute. The last three unresolved sites on the path from resource.rrl.lsu.edu to ozy4.csm.ornl.gov correspond to the firewall at ORNL.

```
[wuq@resource /home/wuq] traceroute ozy4.csm.ornl.gov
traceroute to ozy4.epm.ornl.gov (160.91.77.112), 30 hops max, 38 byte packets
 1  csc-gw.net.lsu.edu (130.39.224.1)  0.375 ms  0.256 ms  0.248 ms
 2  lsubrl.gl.lsu.edu (130.39.1.20)  0.921 ms  1.325 ms  0.705 ms
 3  laNoc-lsubr.LEARN.la.net (162.75.0.9)  1.893 ms  1.911 ms  1.802 ms
 4  abileneHou-laNoc.LEARN.la.net (162.75.0.6)  7.411 ms  8.223 ms  7.675 ms
 5  kscy-hstn.abilene.ucaid.edu (198.32.8.62)  23.505 ms  22.730 ms  23.116 ms
 6  ipls-kscy.abilene.ucaid.edu (198.32.8.6)  33.022 ms  32.632 ms  32.404 ms
 7  chin-ipls.abilene.ucaid.edu (198.32.8.70)  36.238 ms  36.310 ms  35.488 ms
 8  chi-ocl2pos-abilene.es.net (198.125.140.53)  35.947 ms  36.340 ms  36.949 ms
 9  nyc-s-chi.es.net (134.55.205.105)  52.063 ms  62.905 ms  63.691 ms
10  orn-s-nyc.es.net (134.55.205.110)  85.513 ms  84.195 ms  84.195 ms
11  ornl-orn.es.net (134.55.208.62)  104.370 ms  104.732 ms  103.683 ms
12  orgwy.ornl.gov (192.31.96.225)  103.762 ms  103.517 ms  103.535 ms
13  * * *
14  * * *
15  * * *

[wuq@ozy4 /home/wuq] traceroute resource.rrl.lsu.edu
traceroute to resource.rrl.lsu.edu (130.39.224.177), 30 hops max, 38 byte packets
 1  swge6010-07.ens.ornl.gov (160.91.76.1)  0.310 ms  0.385 ms  0.351 ms
 2  swge4500n-64.ens.ornl.gov (160.91.0.9)  4.737 ms  0.429 ms  0.382 ms
 3  ornlgwy.ens.ornl.gov (160.91.0.1)  0.472 ms  0.430 ms  0.395 ms
 4  orgwy-fw.ornl.gov (192.31.96.161)  0.769 ms  0.576 ms  0.544 ms
 5  ornl-rt3-ge.cind.ornl.gov (192.31.96.250)  1.020 ms  0.996 ms  0.566 ms
 6  orn-ornl.es.net (134.55.208.61)  19.580 ms  19.415 ms  19.772 ms
 7  nyc-s-orn.es.net (134.55.205.109)  56.555 ms  52.759 ms  53.479 ms
 8  abilene-nyc.es.net (198.124.216.106)  73.551 ms  73.719 ms  73.672 ms
 9  wash-nycm.abilene.ucaid.edu (198.32.8.45)  78.239 ms  79.417 ms  77.845 ms
10  atla-wash.abilene.ucaid.edu (198.32.8.65)  79.469 ms  79.243 ms  79.473 ms
11  hstn-atla.abilene.ucaid.edu (198.32.8.33)  95.959 ms  95.902 ms  96.424 ms
12  laNoc-abileneHou.LEARN.La.Net (162.75.0.5)  101.848 ms  101.952 ms  101.991 ms
13  lsubr-laNoc.LEARN.la.net (162.75.0.10)  103.545 ms  102.613 ms  102.712 ms
14  dbyd-6509-13-1.gl.lsu.edu (130.39.1.6)  103.700 ms  105.109 ms  104.303 ms
15  resource.rrl.lsu.edu (130.39.224.177)  103.749 ms  106.046 ms  103.491 ms
[wuq@ozy4 /home/wuq]
```

Figure 1.1 Two-directional routes between two hosts located at ORNL and LSU[1]

In a packet-switching network, delays always vary as a result of network state, particularly congestion. Thus, measures of RTT usually give averages, which may have high standard deviations. In general, the busier networks are, the longer packets delay. The relation between packet delay and network congestion will be further discussed later in this Chapter.

RTT is such an important quantity for estimating network condition that it is measured in many transport control schemes. Especially of note, the congestion control of the recently proposed TCP Vegas is largely based on RTT measurements [BP95, MLAW99]. In the design of TCOU, we also estimate RTT and use it for effectively estimating packet loss. Specifically, TCOU assumes that an outbound packet is lost if its acknowledgement is not received within a certain waiting period, which is always set at a value that is multiple times that of the estimated RTT. Depending upon the length of the waiting period, packets that are claimed lost may not be lost but buffered at the intermediate routers or end hosts due to network congestion.

[1] The network path between these two organizations was rerouted in November 2002 to avoid going through New York ESnet, which resulted in a shorter RTT of about 35 milliseconds as opposed to the previous 105 milliseconds before rerouting.

1.2.3 Bandwidth, Channel Capacity, Throughput, and Goodput

Conventionally, people use bandwidth to mean how fast data flows on a given transmission path. In electrical engineering, the term bandwidth refers to the width of a range of electromagnetic frequencies that a signal occupies in a given transmission medium. In the computer networking area, bandwidth is directly proportional to the amount of information that can be passed along a communication channel in a given period of time, so that it bears the unit of bits per second (bps).

Due to the fact that voltages cannot be changed instantaneously and no media is able to conduct currents perfectly, every communication medium has its own limited bandwidth, which is essentially the maximum speed at which the hardware can generate the signal and inject it into the physical wire. Figure 1.2 compares an ideal signal shape with the one in the "real world" when a single bit is transmitted.

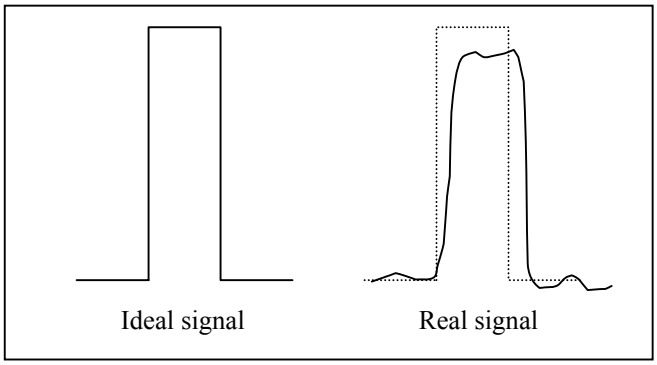


Figure 1.2 Ideal and real voltage change when transmitting a bit of signal

The communication speed has been significantly increased over the past decade. Nowadays, the link bandwidth is in the order of 10 Gbps (OC 192) on the backbone, of hundreds of Mbps to Gbps on Local Area Networks (LANs), and of dozens of Mbps on the last hop to the terminal users using Asymmetric Digital Subscriber Line (ADSL) or cable modems.

The information capacity of channel is defined as the maximum of the mutual information between the channel input X_k and the channel output Y_k over all distributions on the input X_k [Haykin00]. Let B denote the highest frequency of continuous oscillation the communication hardware can generate. The relationship between the signal frequency B and channel capacity C is depicted by the Nyquist Sampling Theorem:

$$C = 2B \log_2 L \tag{1.1}$$

where L is the number of different voltage levels used to represent the information.

If the channel output is perturbed by Additive White Gaussian Noise (AWGN) of zero mean and power spectral density $N_0/2$, the samples of the received signal can be denoted by continuous random variables Y_k:

$$Y_k = X_k + N_k, \quad k = 1, 2, \ldots, K \tag{1.2}$$

where $K=2BT$ is the number of samples in a period of T, X_k is the continuous random variables obtained by uniform sampling of the transmitted signal, and N_k is the noise sample with zero mean and variance given by

$$\sigma^2 = \frac{N_0}{2} \cdot 2B = N_0 B \tag{1.3}$$

The average transmitted power over such a discrete-time, memoryless Gaussian channel is defined as

$$E[X_k{}^2] = P, \quad k = 1, 2, \ldots, K \tag{1.4}$$

Apparently the total received power is the sum of signal and noise power, which is computed as

$$P_{rec} = E[X_k{}^2] + \sigma^2 = P + N_0 B \tag{1.5}$$

Therefore, the corresponding received signal amplitude $A_{signal} = \sqrt{P_{rec}} = \sqrt{P + N_0 B}$ and the noise amplitude $A_{noise} = \sqrt{N_0 B}$. The number of distinguished voltage levels L is the ratio of the signal and noise amplitudes

$$L = \frac{A_{signal}}{A_{noise}} \tag{1.6}$$

Now we may express the capacity of the channel with Gaussian noise disturbance as follows

$$C = 2B \log_2 L = 2B \log_2 \left(\frac{P + N_0 B}{N_0 B} \right)^{1/2} = B \log_2 \left(1 + \frac{P}{N_0 B} \right) \tag{1.7}$$

This is Shannon's famous Information Capacity Theorem, which highlights the interplay among three key system parameters: channel bandwidth, average transmitted power, and noise power spectral density.

Throughput is the "real data transfer rate" at a given time under a certain network condition. It refers to the actual level of end-to-end traffic put through the network across a path between a transmitting device and one or more receiving devices. In practice, a high bandwidth does not necessarily guarantee a high throughput because the achieved throughput does not depend only on the system bandwidth, but maybe more importantly, the efficiency of using the available communication resources. If the efficiency is terrible for sending a megabit, a gigabit line is no better than a megabit line just more expensive [Tanenbaum02].

Another concept closely related to bandwidth and throughput is goodput, which is used in ATM (Asynchronous Transfer Mode) and other packet networks to describe the user payload. As we know, not all network traffic is devoted to user information transfer because a certain (usually relatively small) portion of the capacity is used for other purposes like signaling, synchronizing, and control. Furthermore, the overheads due to various protocol headers and trailers also consume some usable bandwidth. The data packets or cells that make it successfully from end to end contribute to goodput, while the lost ones comprise the badput. Generally, we have the following formula:

bandwidth capacity = goodput + badput + overheads + unused bandwidth

In the context of ATM networks, the throughput is the sum of the goodput and the badput. However, in our work presented in this dissertation, we specifically exclude duplicate and lost packets from calculating the throughput. Therefore, the throughput and the goodput referred to in the context of this dissertation have a similar meaning in terms

of user payload, provided that the length of the user-defined protocol header is much smaller than the size of the useful data portion, which is the case of TCOU.

1.2.4 Delay-Throughput Relation and Bandwidth-Delay Product

Due to many factors, the throughput and delay are not completely independent of each other. Let's take an analogous real-life example of daily transportation on a busy highway. People certainly expect longer driving time to get home from work in rush hour than say at 3:00 o'clock in the morning. The congestion that occurs in computer networks is quite similar to the traffic jam in real life if we liken routers to intersections. Computer researchers came up with the following empirical equation describing the relation between the delay and throughput in a computer network [Comer98]:

$$D = D_0 \frac{1.0}{(1.0 - U)} \tag{1.8}$$

where coefficient D_0 is the delay when the network is light-loaded or empty, and parameter U is a quantified network busy indicator with values between *0* and *1.0* indicating how much the network resources are being utilized, to which the throughput is directly related.

Another important quantity to keep in mind when analyzing the network performance is the Bandwidth-Delay Product (BDP), which is obtained by multiplying the bandwidth in bps by the round-trip time in seconds. This product is the amount of outstanding data that fills up the full-duplex pipe. It is obvious that the receiver's buffer size must be at least as large as the bandwidth-delay product to make the system work at the peak bandwidth.

1.2.5 Packet Loss

The scarcity of network resources is the main reason for causing packet losses during data transmissions over packet-switching networks. Figure 1.3 graphically illustrates such a network with packet loss occurring on its bottleneck link.

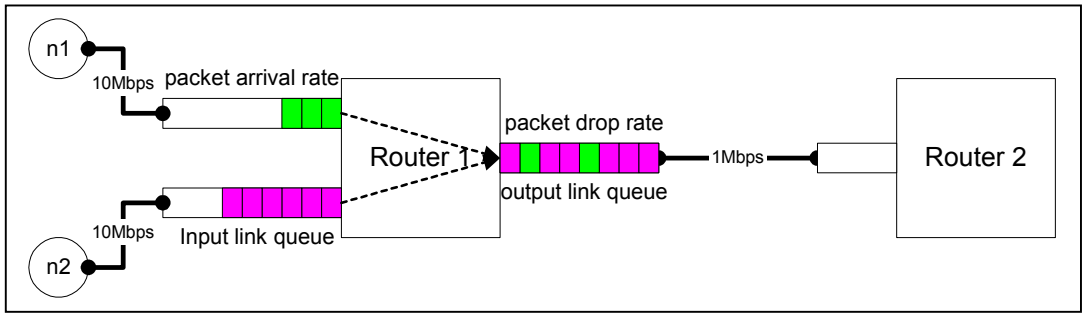


Figure 1.3 Packet loss on a bottleneck link

As shown in Figure 1.3, two nodes are connected with Router 1 by a link of bandwidth 10Mbps, and Router 1 is connected with Router 2 by a link of bandwidth 1Mbps. An incoming or outgoing packet is always buffered for an indefinite period of time waiting for its routing or signaling turn. The routers have a single input/output queue for each attached link. Obviously, the link between Router 1 and Router 2 is the bottleneck link, and packets may be dropped at the output port of Route 1 in times of congestion.

The buffer process in most routers deployed in the Internet is scheduled on a First-In-First-Out (FIFO) basis. FIFO scheduling implements a simple First-Come-First-Served

(FCFS) algorithm without differentiating packets of different flows. For the purpose of fairness, per-flow scheduling mechanisms such as Weighted Round-Robin (WRR) have been proposed to isolate each flow and explicitly control the allocation of bandwidth among a set of competing flows. However, FIFO scheduling is preferable in terms of implementation efficiency, especially when both link speeds and the number of active flows per link increase [FF99].

The queuing discipline employed in a router determines the way of dropping packets in times of network congestion. DropTail, which simply drops an arrival to a full buffer, is the most commonly used one in the present Internet. Random Early Detection (RED) is a type of Active Queue Management (AQM), which maintains an exponentially weighted queue length and drops packets with a probability that increases with the average queue length [LPD02]. The flow's arrival rate could be estimated from the history of packet drops maintained by AQM, and the flow's packet drop rate could be estimated using the aggregate packet drop at the queue [FF99].

Other physical device failures such as power outage, system crash, and link breakdown, may also cause packet losses. However, since the present computer software and network hardware is generally stable for a long period of time, most packet losses are due to network congestion instead of system malfunctions in wired networks. Packet loss rate is also an important measurement of network performance.

It is worth pointing out that the wireless networks have very different link properties than wired networks. The packet losses in a wireless network are very likely to be caused by the signal losses at the physical link level. The current implementation of TCP congestion control is not suited for the wireless environment because of its high packet loss and low probability of simultaneous transmissions at the physical layer. A default TCP sender interprets any packet losses as congestion signals and reduces its sending rate drastically. In the wireless environment, however, the opposite is required: the source rate must be increased to account for packet losses caused by physical wireless link failures [RWIM02].

Having a small mean transmission time and a low packet loss rate is not enough for time-critical applications such as remote medical diagnosis and video-on-demand applications. The sound and image quality may not be acceptable if a serious jitter occurs. Therefore, these applications also demand a small standard deviation of transmission time in order to achieve smooth and reliable aural and visual effects.

1.2.6 Congestion Collapse

Congestion collapse occurs when the network productivity is impaired by an increase in the network load. In the case of congestion collapse, the transmission of overwhelming non-payload data consumes a large fraction of the bandwidth and hence the useful goodput is severely reduced. The non-payload data can be caused by many factors.

The classical congestion collapse is primarily due to unnecessarily retransmitted packets of TCP flows because of the imprecise timer and the defective congestion control mechanism used in early TCP implementations. The congestion collapse that occurs in today's Internet environment mostly results from undelivered packets, which are dropped before reaching their ultimate destination and waste a large amount of bandwidth. The congestion collapse from undelivered packets is primarily caused by the increasing deployment of open-loop applications not using end-to-end congestion control [FF99].

Other forms of congestion collapse include: fragmentation-based congestion collapse, which is in consequence of a mismatch between link level transmission units (cells or frames) and higher layer retransmission units (datagrams or segments); congestion collapse from control traffic, which comes from an increasing load of control traffic on the congested links such as packet headers for small data packets, routing updates, multicast join and prune messages, session messages for reliable multicast sessions, DNS messages, etc.; congestion collapse from stale packets, which occurs if the congested links are busy carrying packets that are no longer wanted by the user [FF99].

1.3 Problem Statement

In this section, we describe the problems associated with the current implementations of transport layer protocols, mainly Transport Control Protocol (TCP), and the problems we attempt to solve in this dissertation.

1.3.1 Problems with TCP

TCP is the most widely used transport protocol in the Internet and carries the majority of the network traffic [WS94, Stevens98]. The Additive Increase Multiplicative Decrease (AIMD) algorithm is the primary mechanism used by the predominantly deployed TCP (Tahoe, Reno) for end-to-end congestion control [Jacobson88, WS94]. Although some researchers believe that AIMD is a necessary condition and is even optimal for a congestion control mechanism to be stable, the recent research work has shown that TCP's AIMD algorithm is also the main cause accounting for many network performance problems. These TCP limitations motivate us to explore a new class of transport protocols with improved congestion control mechanisms.

- **Throughput**

The upper limit of data transmission rate is imposed by the physical media bandwidth. In recent years, bandwidths at all levels of networks have been boosted greatly by the rapid evolution of new technologies and this desire to increase still continues. The current operational wide area networks (WANs), e.g. OC 192 between ORNL and Atlanta, offer 10 Gbps connection speed, and the experimental networks offer link speeds of 100 Gbps in limited scenarios. However, the achieved network throughput depends to a large extent on the protocol performance in the transport layer. A great deal of attention has been paid to the design issues of transport protocols since the early stage of the original network development.

TCP-friendly congestion control, i.e. AIMD algorithm suffers from a primary throughput drawback because TCP interprets every packet loss as a congestion signal and then halves its congestion window, which drastically slows down the transmission. There is a common perception that congestion avoidance is better than congestion recovery in TCP as far as throughput is concerned. However, TCP does not provide any effective ways to specifically avoid congestion. Instead, the TCP congestion window is allowed to increase exponentially in the slow start phase or linearly in the congestion avoidance phase until a packet loss occurs, or the available bandwidth or the receiver's advertised window size is reached. In practice, the two latter cases do not happen as often as the former one.

For a single conformant TCP flow with a packet drop every full window of packets in its ideal equilibrium condition, the TCP sending rate is upper limited by [FF99]:

$$r \le \frac{1.5\sqrt{\frac{2}{3}} \cdot MSS}{RTT \cdot \sqrt{p_{drop}}} \tag{1.9}$$

where p_{drop} is the packet drop rate of the TCP flow. For a 10Mbps path between LSU and ORNL with RTT of 50 ms and MSS of 1460 bytes, the maximum sending rate of TCP is about 2.0Mbps for a packet loss rate of 2%. The goodput of actual TCP connections is generally lower than 1.0Mbps on this path because they may have limited demand, a window size limitation, a smaller packet size, a less-aggressive TCP implementation, a receiver that sends delayed acknowledgments.

- **Dynamics**

Network dynamics is critical for some applications that involve control mechanisms over wide-area networks, such as instrument grids, remotely deployed mobile robot teams, and interactive simulations distributed on supercomputers. The serious jitters that occur during data transmission in the control loop may significantly impair the application performance. Practically speaking, these applications always prefer a slow-but-uniform delivery rate to a fast-but-jumpy one. However, TCP congestion control mechanisms, especially AIMD algorithm, have exhibited complicated end-to-end dynamics over various time scales, which could result in serious negative effects on the controllability and stability of control loops implemented over wide-area networks [RC02]. It is important to understand and control the dynamics of transport layer protocols at the time scales appropriate for the application under study.

- **Fairness**

The packet-switching network is meant to share the network resources among multiple users. An aggressive transport protocol that consumes most of the bandwidth by killing all other concurrently participating flows is obviously undesirable. The fairness objective of TCP is to let each TCP session get *1/N* of link capacity if *N* TCP sessions share the same bottleneck link. Since the congestion window *cwnd* in TCP is adjusted by its self-clocking property: increased by one for every acknowledgement during slow start and every round-trip time during congestion avoidance, decreased to half if a packet loss is detected, a session with shorter round-trip time may suppress a longer ones' throughput [FJ92]. [Floyd91] discusses the relative distribution of bandwidth between two competing TCP connections on paths with different numbers of congested gateways. In the design of the new transport protocols, the fairness problem needs to be taken into consideration.

- **Stability**

Stability is another important issue for dynamic complex systems such as the network transport control system. Theoretically, when a stable point is reached, the window size is converged to a fixed value if the number of connections is not varied and each connection continuously transmits segments or packets. However, due to the chaotic nature of TCP congestion control, the system may enter an equilibrium condition where the fluctuations lead to regular packet drops and result in typical, sawtoothed throughput. This is especially the case for "long fat pipes" with high bandwidth and long delay, where it

takes more time for TCP to reach equilibrium. The factors that impact the stability include round-trip delay, link capacity, traffic load, and active queue management (AQM) [LPD01].

Another protocol implemented in the transport layer is User Datagram Protocol (UDP), which is a simple datagram-based delivery scheme without any mechanisms of flow and congestion control. Since UDP does not guarantee reliable transmission, we will not discuss its implementation details any further. However, it is the simplicity that makes it possible to implement new transport control protocols on UDP. TCOU is one of such UDP-based protocols.

1.3.2 Goodput Stabilization and Maximization Problems

The transport controls for goodput stabilization and maximization are two main problems we attempt to resolve in this dissertation. Here we only briefly present the essence of the problems and leave the detailed problem formulations to their corresponding chapters.

In the goodput stabilization problem, our research objective is to dynamically control the source rate such that the goodput is stabilized at a desired level, which is usually much lower than the maximum achievable goodput level. There are a considerable number of such network applications that require a certain level of goodput to satisfy application performance needs. For these applications, using more bandwidth than necessary is a waste of network resources while insufficient bandwidth allocation cannot guarantee the desirable performance. More importantly, the achieved goodput must remain smooth and stable even in the presence of various background traffics. Apparently, the current implementation of TCP is not able to handle the goodput stabilization problem because TCP has no knowledge of the throughput needs of individual users. Furthermore, TCP's AIMD congestion control algorithm makes it very difficult to maintain the goodput at a smooth and stable level.

In the goodput maximization problem, we aim to dynamically control the source rate to achieve high bandwidth utilization by fairly maximizing the individual throughputs from the overall perspective. A great amount of network research efforts have been focused on the goodput maximization problem, which actually identifies two considerations: fair share and high utilization of bandwidth. Fair share requires that all concurrent data streams sharing the same link be treated equally, and high utilization requires that the link bandwidth is exploited to the greatest possible advantage. TCP's congestion control mechanism makes TCP a nice neighbor to other concurrent TCP sessions, but its conservative nature also unfavorably results in low efficiency of bandwidth usage. In most cases, the throughput achieved by TCP only occupies a very small portion of the channel capacity. At the other extreme, some aggressive transport protocols like Tsunami may achieve high bandwidth utilization by unfairly maximizing the individual throughput. These protocols unavoidably cause serious fairness problems because they occupy most of the available bandwidth and suppress all other concurrent TCP-friendly data streams.

The difficulty of transport control for good stabilization and maximization essentially arises from various types of randomness involved in the end-to-end delay and throughput measurements during data transmission over wide-area networks, which account for the stochastic nature of network traffic. This is the main motivation underlying our research in developing a new set of transport control protocols based on stochastic approximation methods.

1.3.3 Transport Control Problems in Wireless Networks

As we know, wireless networks have very different link characteristics from wired networks. Here we address two problems associated with transport control in wireless networks.

Firstly, the AIMD congestion control algorithm in default TCP is not well suited for a wireless environment because packet losses in the wireless network do not necessarily indicate network congestion. The packet losses over a wireless link are mostly due to physical link failures. However, TCP always interprets every packet loss as a signal of network congestion and then immediately halves its congestion window to reduce the source rate drastically. Therefore, the AIMD algorithm makes the poor performance of TCP throughput even worse in a wireless network. In the case of packet loss caused by physical link failures, instead of reducing the sending rate, we need to increase the source rate accordingly to account for lost packets.

The second problem in the wireless network is related to the transmission connectivity issue. In an ad-hoc wireless network deployed in an unstructured area, specially designed routing facilities are lacking and the network topology is subject to dynamic change due to the node movements. The default TCP needs the support of underlying routers and requires a path connection existing between source and destination nodes during the whole period of data transmission. The routing and connectivity requirement is always satisfied in the Internet but usually not guaranteed in an ad-hoc wireless network. However, data is still deliverable between nodes that are never even connected at any time if we take advantage of the movements of intermediate nodes. This is the motivation for us to develop a set of wireless-specific protocols for transport control to resolve the routing and connectivity issues in wireless networks.

1.4 Main Approaches

1.4.1 ONTCOU

Determining a proper source rate online to meet different application performance requirements in the presence of dynamic background trafficking is the key to the design of good transport control protocols. In light of the problems associated with the default TCP and UDP in the transport layer, and the difficulties of the goodput stabilization and maximization problems we attempt to solve, we shall develop a new class of protocols, referred to as Transport Control Over UDP (TCOU), which employs dynamic stochastic approximation methods to adjust the source rate.

The developmental work of TCOU is realized using overlay networks of NetLets daemons [Rao01], which reside at the application level and perform transport controls via UDP. By building TCOU on the Overlay Network, we propose a complete implementation framework, named Overlay Network with Transport Controls Over UDP (ONTCOU) to overcome the limitations of default TCP and UDP in the aspects of throughput, stability, dynamics, and fairness. The relationship between the proposed solution of ONTCOU and other research areas is depicted in Figure 1.4.

To meet the challenges brought about by wireless link characteristics, we also make efforts to adapt ONTCOU to wireless network environments with regard to transmission and connectivity issues.

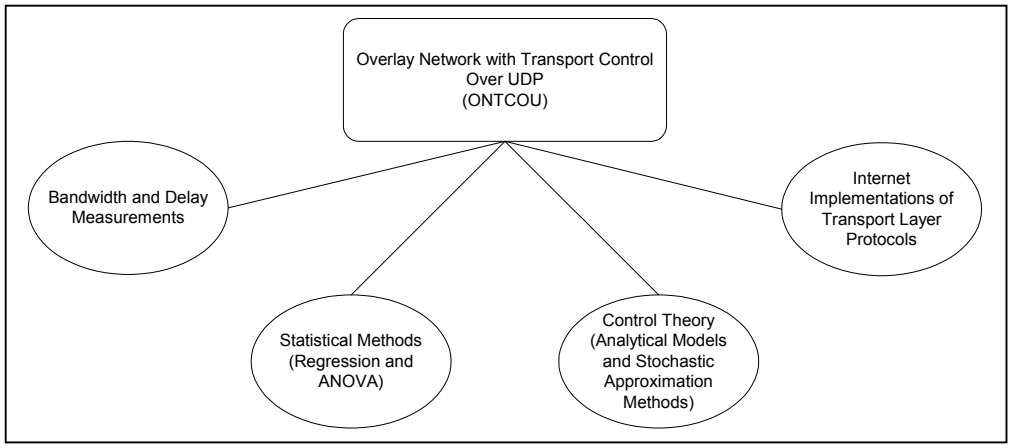


Figure 1.4 Relationship between ONTCOU and other research areas

1.4.2 Stochastic Approximation Methods

The stochastic approximation (SA) methods have been particularly useful in solving for the root of an equation or the minimum (or maximum) of a function in the presence of randomness in a complex system. The randomness may be present as either noise in measurements or Monte Carlo randomness in the search procedure, or both. Many different types of SA methods have been developed to solve problems in different fields. The most frequently used SA methods include Robbins-Monro SA, Finite Difference SA, Kiefer-Wolfowitz SA, Random Direction SA, Simultaneous Perturbation SA, etc.

There are various noise components involved in network performance measurements. The stochastic nature of network traffic necessitates leading the stochastic approximation methods into the design of transport control protocols. The main advantage with the stochastic approximation methods is that they require nothing more than the output estimation, which is easily available in practice, to control the system [WASAN69]. Furthermore, as a non-parametric technique, it is unnecessary to know the exact form of the regression function or to estimate any unknown parameters in the SA methods.

My personal understanding of applying SA methods to transport protocol design is to "Let nature take its course" to avoid using flow and congestion control parameters, whose values are usually subjectively pre-selected by protocol implementers.

1.5 Dissertation Structure

The rest of the dissertation is organized as follows:

In Chapter 2, we conduct a broad survey of the research background and present an overview of related work with focus on the techniques used in the current implementation of widely deployed TCP.

In Chapter 3, we design a basic transport control model without congestion control mechanism and collect extensive network traffic measurements, based on which, a two-way statistical analysis is performed to determine the main effects of two source rate control parameters and the interaction effects between them.

In Chapter 4 and Chapter 5, we develop and design a set of transport control protocols using stochastic approximation methods for goodput stabilization and maximization with the focus on the convergence proof of the methods and the protocol implementation

details. Many experimental results are presented to justify the effectiveness of the new transport control protocols.
In Chapter 6, we develop an overlay network of NetLets, which perform link bandwidth measurements and carry out multiple quickest paths computing and routing. The overlay network of NetLets together with the transport control over UDP using stochastic approximation methods forms our proposed research base, ONTCOU.
In Chapter 7, we introduce a Connectivity-Through-Time concept and adapt ONTCOU to the wireless environment by developing and integrating a set of wireless-specific protocols for transport control in ad-hoc wireless networks.
Finally in Chapter 8, we give a summary of the dissertation and present extensions and ideas for future work.

1.6 Main Contributions of the Dissertation

(1) We investigate the stochastic nature of network traffic and design a novel model for source rate control. We collect extensive network performance measurements and apply statistical analysis methods to determine the main effects and interaction effects of two parameters in the transport control model.
(2) We develop and implement a new class of end-to-end transport control mechanisms based on stochastic approximation methods to solve the goodput stabilization and maximization problems. We mathematically show that these methods converge nicely under relatively loose conditions, which is further justified by extensive experimental results. We stabilize the goodput smoothly at a certain desired level and achieve the maximum throughput consistently three times more than the throughput achieved by default TCP. The fairness problem is also implicitly handled by the dynamic version of the selected SA methods. These stochastic approximation method-based protocols are expected to improve or even replace the current TCP design methodology.
(3) We develop and apply new techniques in the implementation of TCOU, such as floating window based flow control, RTT based packet loss detection, new congestion recovery, etc.
(4) We define fundamental structures and construct a complete implementation framework of ONTCOU to overcome limitations of default TCP and UDP in the aspects of throughput, fairness, stability, and dynamics. We implement bandwidth and delay measurements as well as multiple quickest paths algorithm in the overlay network of NetLets to guarantee end-to-end delay. The implementation of overlay networks makes it possible to set up a bridge between theoretical research and Internet deployment of new transport control protocols.
(5) We adapt ONTCOU to a wireless network environment with lossy wireless link characteristics and resolve the wireless-specific congestion control and connectivity issues.

CHAPTER 2 BACKGROUND SURVEY AND RELATED WORK

2.1 Network Protocols and Layering Models

The network protocol is a standard procedure for facilitating data transmission between nodes geographically distributed in networks. Figure 2.1 illustrates the importance of establishing a set of standard network protocols for computer communications.

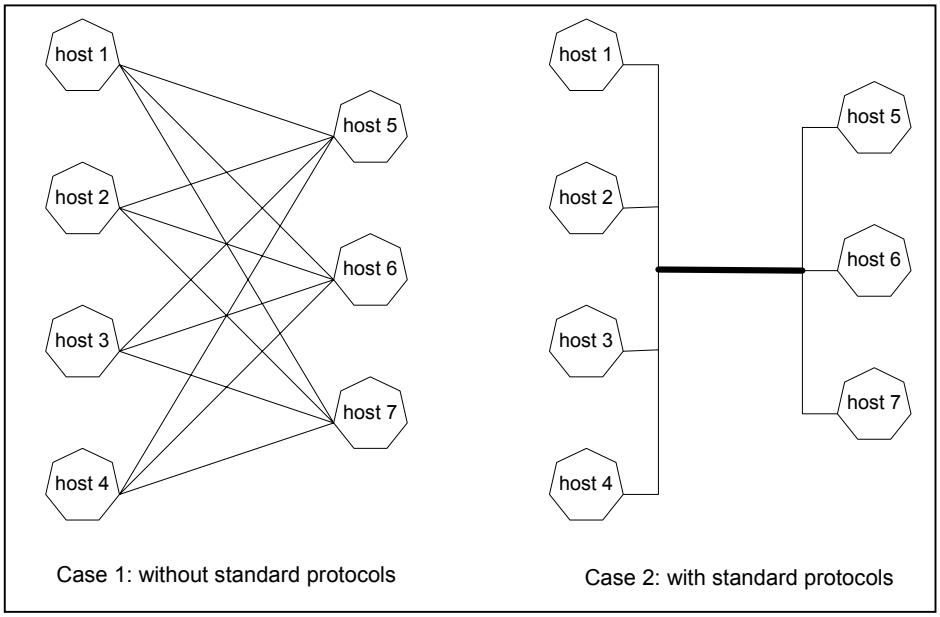


Figure 2.1 Communications between computers with and without standard protocols

In Figure 2.1, each communication link requires a set of rules regulating the "talk" between two end hosts. Apparently, every host must master 3 or 4 "languages" to communicate with others in Case 1 while only one is needed in Case 2. As an analogous example in the community of human beings, a universal language might help us cast off the burden of learning multiple foreign languages.

Protocol layering is essential to reduce design complexity and enable graceful extensions in the future. The *International Organization for Standardization* (ISO) has created a *Reference Model of Open System Interconnection* (OSI), which consists of seven layers as tabulated in Table 2.1 [Tanenbaum02]. The OSI reference model deals with connecting systems that are open for communication with other systems.

Table 2.1 The OSI reference model

Layer 7	Application	Message passing
Layer 6	Presentation	Encoding
Layer 5	Session	Authentication and encryption
Layer 4	Transport	Segments and datagrams
Layer 3	Network	Packets
Layer 2	Data Link	Frames
Layer 1	Physical Hardware	Signaling and wiring

TCP/IP Reference Model is the most widely employed model in all computer networks, from the ARPANET to the worldwide Internet. It was first defined in [CK74], and the design philosophy behind the model was further discussed in [Clark88]. Figure 2.2 shows the architecture of TCP/IP reference model with Transmission Control Protocol (TCP) instantiated in the transport layer as well as the data flow in the network system. The TCP/IP model shown in Figure 2.2 has neither session nor presentation layers because these two layers are of little use to most applications.

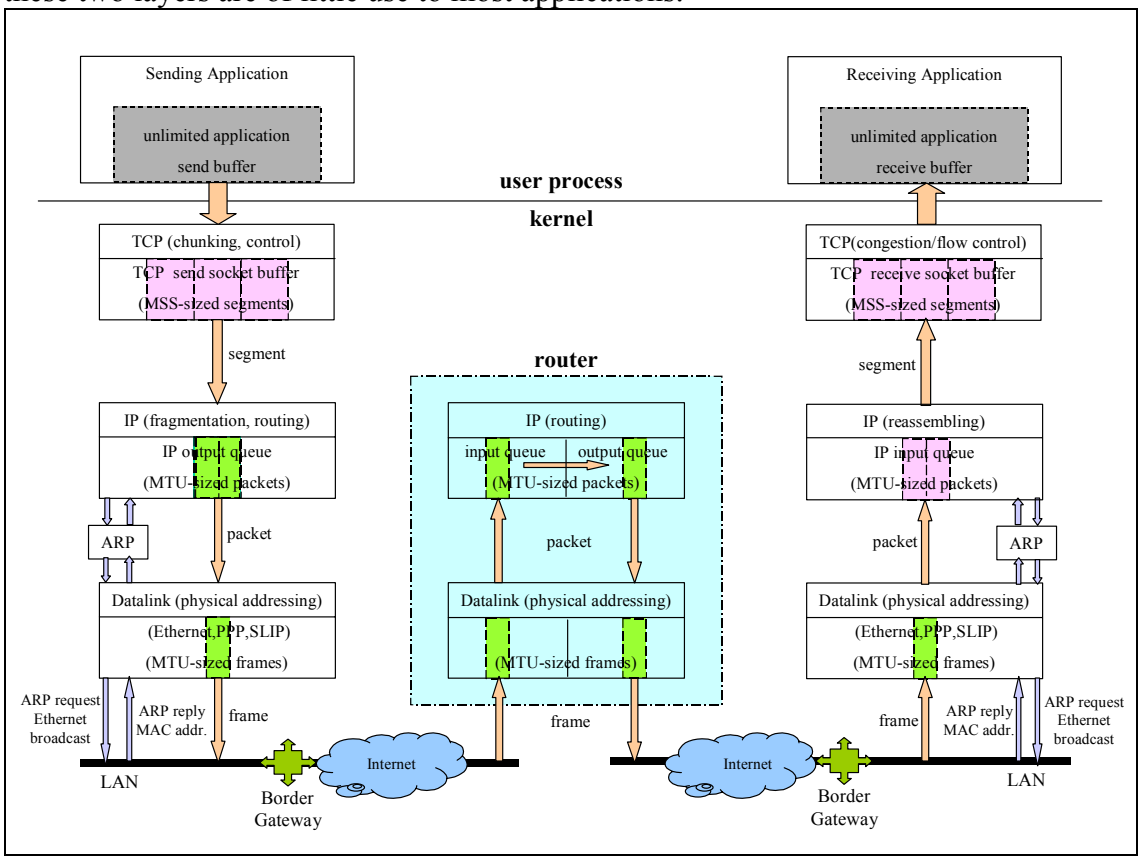


Figure 2.2 Data flow in TCP/IP stack

As illustrated in Figure 2.2, calling a *write()* function makes the kernel copy all the application data from the application buffer to the TCP send buffer where the data is chunked into segments of Maximum Segment Size (MSS). The kernel will not return from the *write()* function until the final byte in the application buffer has been copied into the TCP send buffer. TCP sends the data in MSS-sized chunks or smaller to Internet Protocol (IP) layer after prepending its TCP header to each segment. For a UDP socket, the successful return from a *sendto()* function indicates that either the datagram or all fragments of the datagram have been added to the datalink output queue. There is no actual send or receive buffer inside UDP because UDP does not guarantee reliable transmission so that the outstanding data does not need to be buffered.

A segment or datagram sent down by the overlying TCP or UDP is further fragmented into smaller packets at the IP layer if the segment or datagram size is bigger than the Maximum Transmission Unit (MTU) of the underlying datalink layer. The main task of IP is to search the routing table for the next-hop IP address leading to the destination IP address, determine the outgoing interface, and forward each packet to the appropriate

datalink after prepending its IP header. From the next-hop IP address, IP also acquires the next-hop Ethernet MAC address through Address Resolution Protocol (ARP) by broadcasting the ARP request over its directly connected network.
The datalink layer deals with both hardware and software controls of data transmission. MSS-sized frames of data are placed in the output queue of datalink and wait for the availability of the physical transmission medium. The bottom physical layer is responsible for signaling and wiring to deliver the stream of bits to the next hop.
In the intermediate routers, the activities of the lowest three layers (Physical, Datalink, and IP) are repeated so that the data is able to move from the source to destination nodes hop by hop. All layers at the receiver side perform the same tasks as the sender side but in an opposite way and reverse order.

2.2 Protocols in Transport Layer

The base function of the transport layer in the network protocol stack is to provide data transferring service to the user applications. The OSI and TCP/IP layering model defines two protocols at the transport layer: Transport Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a reliable, connection-oriented, byte-stream-based protocol; while UDP is a simpler protocol providing unreliable, connectionless, datagram-based delivery [Stevens98].
TCP [Postel81, APS99, Braden89, Jacobson88] uses a variety of mechanisms to guarantee reliable transmission and attempts to achieve fair sharing of network resources among users, such as byte sequencing, positive acknowledgement, lost packet retransmission, congestion avoidance/recovery, and sliding-window based flow control. The TCP congestion control based on Additive Increase and Multiplicative Decrease (AIMD) algorithm unfavorably imposes a major methodological limit on transmission throughput and results in complicated end-to-end dynamics. TCP performance might not appropriately scale when both bandwidths and delays are rapidly increased, which is the trend of the present network evolution. The problem of running TCP over wireless networks is even much worse because most of packet losses in the wireless environment are not caused by network congestion but physical link failures.
The unit of data processed by UDP is usually called datagram. UDP offers a direct way to send and receive datagrams over networks, which makes it much faster than TCP in terms of transmission rate, but its performance suffers from transmission errors and unreliability in either one of these situations: packet lost, packet partially damaged, packet delivered out of order, and packet duplicated.
Both TCP and UDP use Internet Protocol (IP) for packet delivery. Once a packet is sent over the network through either TCP or UDP, it is most likely to experience an extremely intricate delivery process before it reaches the destination. Very little control can be exerted at the intermediate routers for a packet. But specially designed network instruments at the end points may collect useful bandwidth and delay measurements to estimate network conditions. Based on these measurements, multiple diverse paths can be employed and the packet delivery can be rerouted at the higher layer to offer considerable performance advantages over default TCP or UDP connection. This is the basic idea of overlay routers and networks [Rao01].

2.2.1 Historical Background

The de facto standard for TCP/IP implementations has originated from the Computer Systems Research Group at the University of California at Berkeley, and was first distributed with the 4.2BSD system in 1983. Ever since, the performance of TCP/IP protocol stack has been improved in each of its new distributions.

Particularly, significant features like slow start, congestion avoidance, and fast retransmit were implemented in 4.3BSD Tahoe, which is released in 1988. Afterwards, fast recovery was added to 4.3BSD Reno released in 1990 and it works with fast retransmit to avoid the need for slow start after a single packet loss thereby preventing the pipe from going empty after fast retransmit. A modification of Reno leads to new-Reno TCP, which can recover without waiting for retransmission timer to expire in the case of multiple packet drops. SACK TCP is a conservative extension of Reno TCP, which uses SACK option to report non-contiguous blocks of data that has been received and queued at the receiver. TCP Vegas [BP95] checks timeout on receiving the first duplicate acknowledgement instead of waiting for the third duplicate acknowledgement to speed up the packet loss detection. The congestion window size during the slow start phase in TCP Vegas is increased more cautiously. To make more efficient use of the available bandwidth, TCP Vegas also applies an improved congestion avoidance mechanism, which uses queuing delay as a measure of congestion and adjusts its sending rate proportionally to the ratio of round-trip propagation delay to queuing delay. The various BSD releases up to 4.4BSD-Lite was shown in Figure 2.3, which is adapted directly from [Stevens94].

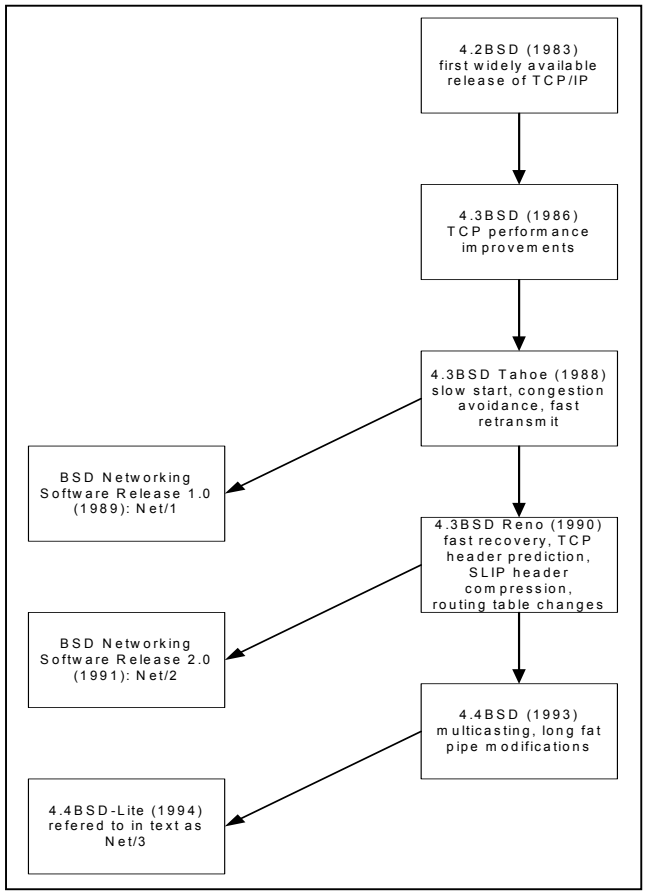


Figure 2.3 A history of BSD releases with key TCP features

It is well known that Linux has overwhelmingly attracted attention from users, developers, and hackers since it came to the free software world. Initially the major reason of success for Linux could be that Linux provided source code while BSD systems did not. The increasing popularity of Linux makes it even harder for the BSD systems (FreeBSD, NetBSD, and OpenBSD) to grow as popular as Linux. There is an interesting statement in [WebDarwin], which says "History shows that first, even if worst, tends to gain power and hold on to it. This is true for Microsoft's dominance of the commercial and home desktop; it's true for Unix/Linux's dominance as the engine for Internet servers; and it's true for Linux's dominance of the freeware OS niche." Most of our implementations will be conducted on Linux machines as to be mentioned somewhere later in this dissertation. However, the developed software is socket-based and can be easily transported to other OS's so that we do not require kernel modifications unlike web100 and net100 instruments [Web100, Net100].

2.2.2 TCP Transmission Control Dynamics

Flow and congestion control are two primary mechanisms adopted by TCP to ensure reliable and effective data transmission. Flow control is intended to avoid overloading the network by properly adjusting the source rate, while congestion control is used to resolve network overload by fairly discarding packets in event of congestion and re-transmitting them later.

2.2.2.1 Byte Sequencing and Cumulative Acknowledging

Sequencing and acknowledging are two basic strategies used by TCP to achieve transmission reliability. Byte sequencing assigns a sequence number to every single byte in the data portion of a TCP segment so that it can always be identified as a range of continuous numbers. However, there are exceptions for two types of segments SYN and FIN, which will be discussed later.

When a segment successfully arrives at the destination, the receiver TCP is required to send an acknowledgement (ACK) bearing the very next expected sequence number back to the sender. In the equilibrium condition, each arriving ACK triggers a transmission of a new segment, which is an important property of TCP, called *self-clocking*. In most of the TCP implementations, the delayed ACK scheme makes TCP not acknowledge a received segment immediately, but wait for a certain time during which the ACK may be "piggybacked" with the data segment going out in the same direction as ACK if any. It was noted in [APS99, Braden89] that TCP must not delay acknowledgements for more than half a second and should send an acknowledgement for every second received segment. In practice, a 200-ms delay is used in most TCP implementations, that is, TCP will delay an ACK up to 200 ms to see if there is any data to send with the ACK [Stevens94]. The effect of extended ACK intervals on TCP connection throughput was also studied in [Johnson95].

TCP uses a cumulative acknowledgement scheme in which only the very next expected segment (i.e. the one pointed by the system variable *snd_una* at the TCP sender) is acknowledged, while all other received segments that are not at the left edge of the receive window are not acknowledged. From cumulative contiguous acknowledgements, a TCP sender can only learn about a single lost packet per round trip time. A Selective Acknowledgement (SACK) mechanism, combined with a selective repeat retransmission policy was proposed in [MMFR96] to deal with multiple packet losses from one window

of data. The receiving TCP sends packets with SACK options to the sender informing the sender of the blocks of data that have been received. Thus the sender only needs to retransmit the missing data segments to fill the holes in the window.

2.2.2.2 Sliding Window

TCP provides flow control through the "*sliding window*" technique. To prevent a fast sender from overflowing a slow receiver, TCP always tells its peer exactly how many bytes of data it is willing to accept from the peer. This is called the *advertised window* [Stevens98]. The receiver window (*rwin*) is the window most recently advertised by the receiver. The sender TCP also maintains another important state variable – *congestion window* (*cwin*) to control its sending rate together with the *advertised window*.
At any given time, the sender must not send data with a sequence number higher than the sum of the highest acknowledged sequence number (*snd_una*), which indicates that all segments less than this sequence number have been received, and the minimum of *cwnd* and *rwnd* [APS99]. In other words, the sender can only transmit an amount of data up to the minimum of the *congestion window* and the *advertised window*. The *congestion window* is flow control imposed by the sender, while the advertised window is flow control imposed by the receiver [Stevens94]. In Reno and NewReno TCP implementations, *cwnd* is used as an offset from *snd_una*, which is a TCP state variable naming the next byte of data to be acknowledged. The sender TCP uses the state variable *snd_nxt* to name the next byte of data to be transmitted. If *snd_nxt* equals *snd_una*, then there is no outstanding data in the sender buffer, that is to say, all previous packets have been acknowledged. A typical sliding window is visualized in Figure 2.4 with key state variables labeled.

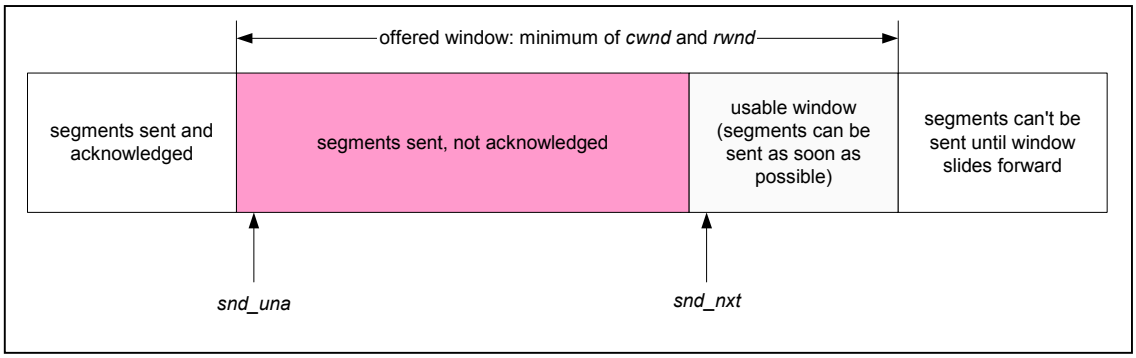


Figure 2.4 TCP sliding window

The sliding window *closes* if the left edge advances to the right when the outstanding data is acknowledged. The window *opens* if the right edge moves forward when the receiver frees up more space and advertises a larger window size or the sender *congestion window* increases. The *shrinking* of window by moving the right edge to the left is usually not recommended [Stevens94]. In no case the left edge will move to the left because it is meaningless to wait for an acknowledgement, which has been received. The segments in the usable window can be sent as soon as the transmitting hardware resources are available.

2.2.2.3 Slow Start, Fast Retransmission, and Congestion Avoidance

Slow start, fast retransmission, and congestion avoidance were added in 4.3BSD Tahoe implementation released in 1988 to improve TCP performance. In view of the difficulty of assessing and monitoring the network condition, TCP starts off a byte-stream transmission in a conservative way: the initial window (IW), the size of the sender's *congestion window* after the three-way handshake is completed, is set one segment. Then, for every received acknowledgement, the *congestion window* is increased by two segments until a segment loss is detected or the *congestion window* reaches the slow-start threshold (*sshresh*). The fast retransmission scheme considers three duplicate acknowledgements (DUPACKs) as an alternative congestion signal to avoid waiting for a retransmission timeout.

When a congestion indicated by the reception of three DUPACKs occurs, the current offered window (the minimum of *cwnd* and *rwnd*) is halved and stored as a new value of *ssthresh*. However, if the congestion is indicated by a retransmission timeout, *cwnd* is set to one segment to repeat over a slow start as the beginning.

Every time when a new acknowledgement arrives, *cwnd* is increased. The way of increasing *cwnd* depends on whether it is in slow start or congestion avoidance phase. If *cwnd* is less than *ssthresh*, slow start is performed and *cwnd* is exponentially increased; otherwise, TCP enters congestion avoidance phase and *cwnd* is incremented additively by 1 full-sized segment per round-trip time. One formula commonly used to update *cwnd* during congestion avoidance is given as

$$cwnd(new) = cwnd(old) + SMSS * SMSS / cwnd(old) \tag{1.9}$$

where SMSS represents the sender maximum segment size. Each time when an acknowledgement is received, the adjustment is performed using Equation (1.9). For a connection in which every segment is acknowledged, Equation (1.9) is slightly more aggressive than one segment per RTT, while for a connection in which the receiver acknowledges every other segment, Equation (1.9) is less aggressive [APS99].

All control phases discussed above that a TCP may experience during transmission are shown in Figure 2.5, which is modified from [Balakrishnan98].

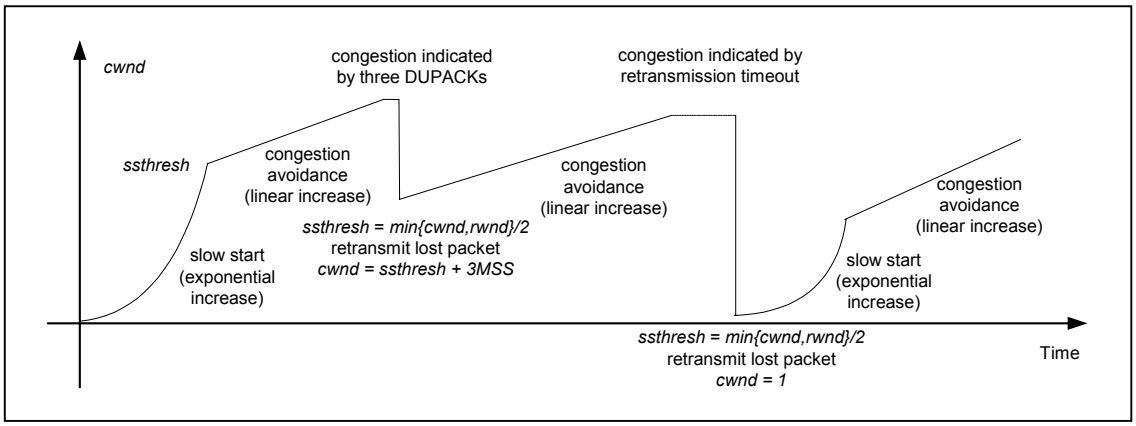


Figure 2.5 Various phases in TCP congestion control

2.2.2.4 Fast Recovery

Fast recovery scheme was implemented in 4.3BSD Reno (1990). Instead of halving *cwnd* immediately after congestion is detected and keeping the *cwnd* size frozen at the same level during the lost packet retransmission, fast recovery allows *cwnd* to be continuously increased upon the reception of each additional DUPACK until the retransmitted segment is acknowledged.

The fast recovery algorithm treats each additional arriving DUPACK as an indication that a segment has left the network (buffered by receiver TCP and waiting for holes to be filled) and released the network resources it occupied, which makes it reasonable to inflate the congestion window by one MSS per each DUPACK. After the congestion window is inflated big enough to have usable window, that is, *cwnd* is bigger than the size of current outstanding segments, each DUPACK triggers a transmission of a new segment. TCP exits fast recovery by reducing its *cwnd* to *(ssthresh+MSS)* when a non-duplicate acknowledgement (i.e. the acknowledgement of the retransmitted segment) arrives. This acknowledgement usually cumulatively acknowledges a large number of segments, which are sent between the first transmission and retransmission of the lost segment, so that the sender is able to transmit a burst of segments at that time. Figure 2.6 shows the *cwnd* variation vs. time during congestion recovery.

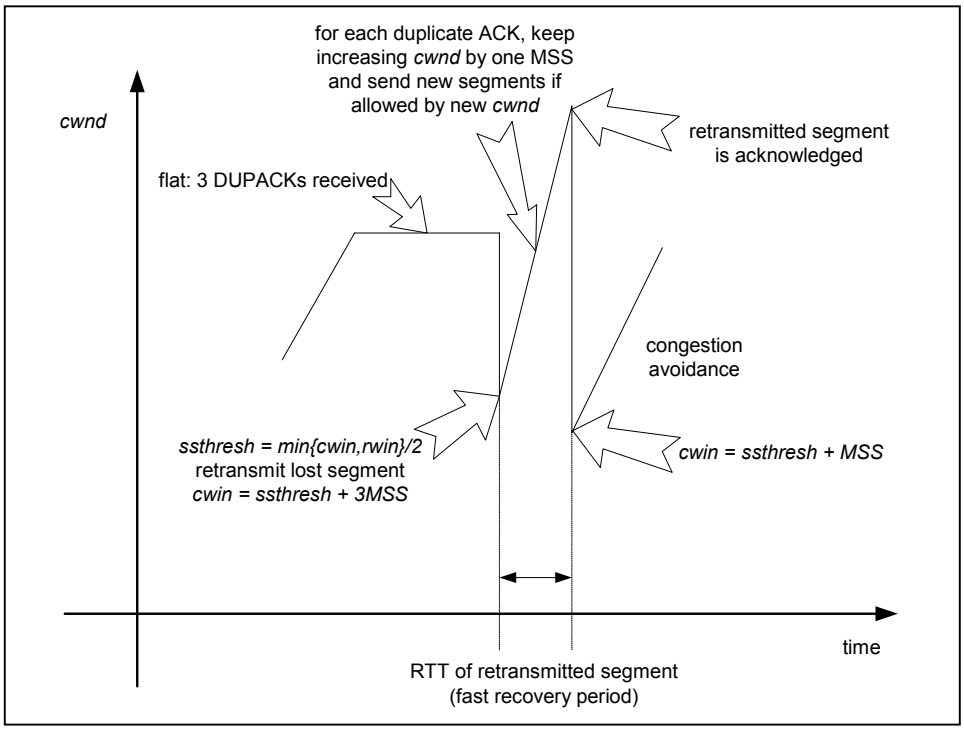


Figure 2.6 Congestion recovery

2.2.2.5 Additive Increase Multiplicative Decrease

TCP uses its end-to-end congestion control mechanisms to achieve robustness over the current Internet [Jacobson88]. Since the congestion window is increased linearly in congestion avoidance phase while decreased to almost half of the original value when congestion occurs, this congestion control is often referred to as Additive Increase

Multiplicative Decrease (AIMD) algorithm. Many new congestion controls proposed to address the needs of new multimedia applications recently [CPW98, JE96, MF97, YL00, PKTK99] are also based on the AIMD method. It is even commonly believed that AIMD is optimal and is a necessary condition for congestion control mechanism to be stable [PD99]. AIMD was first studied in [CJ89] using a system model with a feedback loop as Figure 2.7 shows.

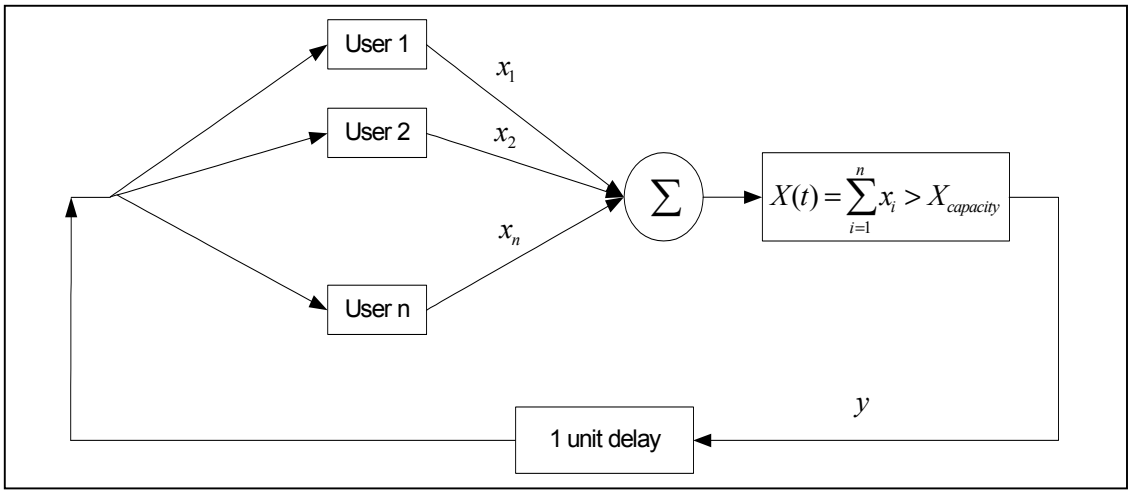


Figure 2.7 Congestion system model

The *i*-th user generates a certain amount of load represented by x_i. The feedback y is determined by comparing the sum of the loads to the load capacity $X_{capacity}$. The dynamics of the system can be formulated by

$$x_i(t+1)=\begin{cases} a_I + b_I x_i(t) & \text{if } y(t)=0 \text{ when } X(t) \le X_{capacity} \\ a_D + b_D x_i(t) & \text{if } y(t)=1 \text{ when } X(t) > X_{capacity} \end{cases} \tag{1.10}$$

The following parameter selection for different feedback statuses specifically determines an AIMD strategy:

$$\begin{cases} Increase: a_I > 0, b_I = 1 \\ Decrease: a_D = 0, 0 \le b_D < 1 \end{cases} \tag{1.11}$$

However, there is a slight difference between the congestion system model illustrated in Figure 2.7 and the AIMD model used in TCP, which decrements the congestion window only for those TCP sessions that detect packet loss or retransmission timeout (RTO) situations when congestion occurs.

The chaotic nature of TCP congestion control is discussed in [VB00], which demonstrates the major features of chaotic systems in TCP/IP networks, like unpredictability, odd periodicity, and extreme sensitivity to initial conditions. In [RC02], an analytical model is presented to analyze the dynamics of a simplified version of TCP, which consists of two unstable linear-like regimes with state space characterized using congestion window size, end-to-end packet delay, and the number of packet retransmissions and acknowledgements. Problems of fairness and steadiness in AIMD are also investigated in [YKZL00] as well as the chaotic behavior of AIMD congestion control.

2.2.2.6 Three-Way Handshake and Four-Segments Termination

Three-way handshake is a mechanism used by TCP to initiate a new connection between two ends. The idea is simple and similar to the regular phone call. The conversation initiator (active end) calls *connect()* function, which sends "can you hear me?" message, a synchronization segment (SYN) with sequence number *J* to the other end (passive end). Upon the reception of this probe, the passive end responds with a message "Yes, I hear you. Can you hear me?", a synchronization segment with sequence number *K* and acknowledgement number *J+1*. When this message arrives at the active end successfully, an ACK of number *K+1* will be sent by the active end to inform the passive end that "Yes, I hear you". The three-way handshake completes when this last ACK segment reaches the passive end. It has been proved that the three-way handshake is the minimum number of steps to create a reliable connection between two ends.

To terminate a TCP connection, the active end sends a finish segment (FIN) with sequence number *M* to the passive end by calling *close()* function. The passive end replied with an ACK of number *M+1*. Sometime later when the passive end completes its work, it will also calls *close()* function to terminate the connection from its own side to the other side. In some cases, not all four segments are needed to terminate a TCP connection. A closed socket cannot be used to either receive or send data. Another socket API function *Shutdown()* may be used to realize half-close (read only or write only) of a socket.

Three-way handshake and four-segments termination are both illustrated in Figure 2.8 [Stevens98]. Although SYN and FIN do not carry any user application data, each of them occupies one sequence number. That is why the responded ACK always bear a sequence number one bigger than the sequence number of SYN or FIN.

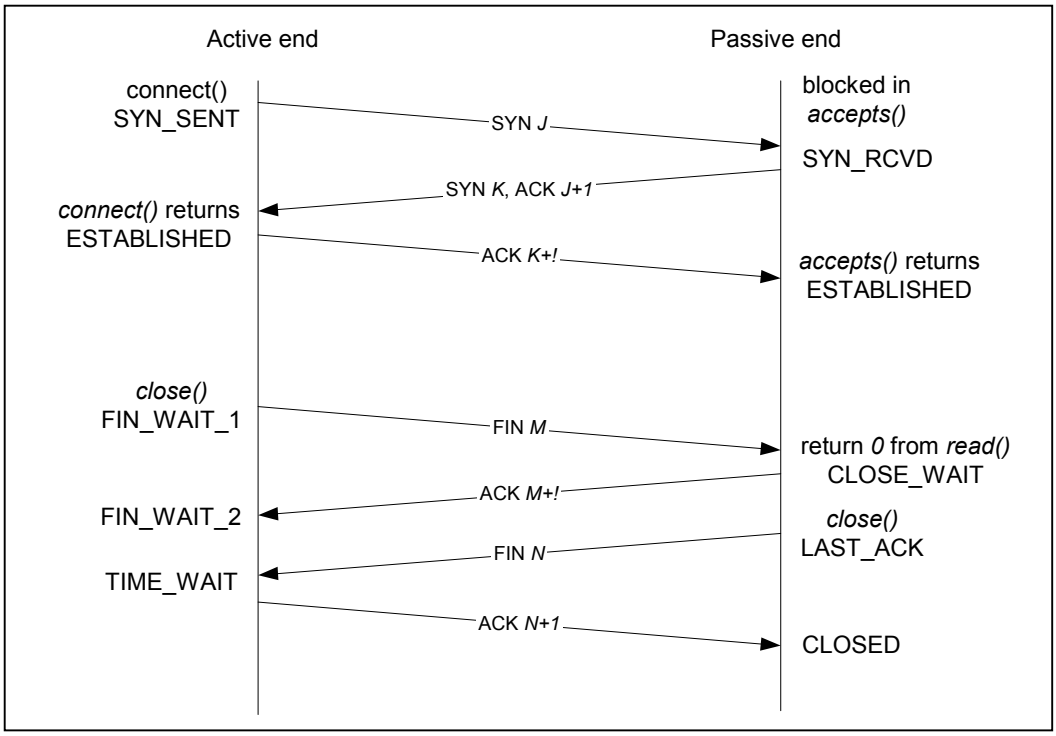


Figure 2.8 TCP three-way handshake and four-segment termination

2.2.3 Mathematical Modeling of Internet Congestion Control

Besides the Internet community that is devoted to the design and implementation of network protocols, the theoretical researchers have been exploring the network traffic nature by analytically modeling the Internet using mathematical tools. In recent years, many analytical models for Internet congestion control have been proposed [KMT98, LL99, MLAW99, KS00].

An optimization-based framework was described in [LPD02] with the intention of analyzing various congestion control mechanisms for a network with N sources and L links, which is generally modeled in Figure 2.9.

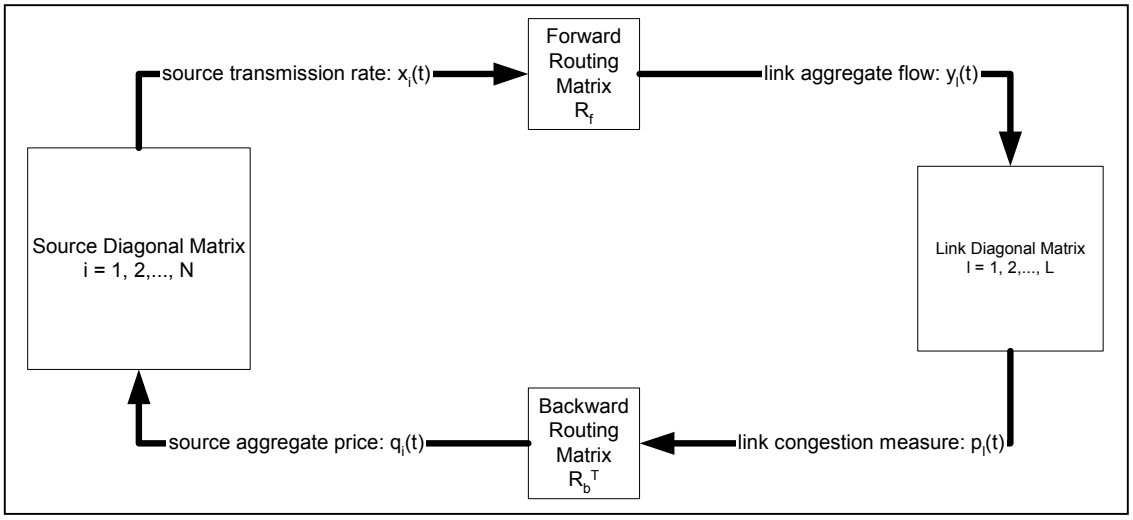


Figure 2.9 A general congestion control model

There might be some forward delays from sources to links and backward delays from congestion measures to source prices in the feedback path. Decentralization is the key restriction in the control laws depicted in Figure 2.9. Each source or link does not have global knowledge, but local information. The relations between control and measure parameters in the congestion control model can be shown mathematically in the form of vector and matrix:

$$\overbrace{[x_1\ x_2\ \cdots\ x_i\ \cdots\ x_N]}^{N\ sources}\cdot\overbrace{\begin{bmatrix}1\,0\cdots1\cdots1\\0\,1\cdots1\cdots0\\\cdots\cdots\cdots\\1\,1\cdots0\cdots1\end{bmatrix}}^{L\ links}=\begin{bmatrix}y_1\\y_2\\\cdots\\y_l\\\cdots\\y_L\end{bmatrix}\xrightarrow{\text{pricing scheme}}\overbrace{[p_1\ p_2\cdots p_l\cdots p_L]}^{L\ links}\cdot\overbrace{\begin{bmatrix}1\,0\cdots1\cdots1\\0\,1\cdots1\cdots0\\\cdots\cdots\cdots\\1\,0\cdots0\cdots1\end{bmatrix}}^{N\ soures}=\begin{bmatrix}q_1\\q_2\\\cdots\\q_i\\\cdots\\q_N\end{bmatrix}$$

The different queue management strategies used in TCP results in different TCP versions: A Vegas source uses queuing delay as a congestion measurement which is updated by the First-In-First-Out (FIFO) buffer process, while a Reno source uses packet loss as a congestion measurement which is typically generated through DropTail or Random Early Detection (RED) queuing discipline [FJ93]. The source control, link control, and utility function in the analytical models for Reno and Vegas are tabulated in Table 2.2 for comparison.

Table 2.2 Analytical models of congestion control for Reno and Vegas [LPD02, LPW02]

TCP version	Analytical Model of Congestion Control	
Reno (RED)	Source sending rate control	$w_i(t+1) = w_i(t) + \frac{x_i(t)(1-q_i(t))}{w_i(t)} - x_i(t)q_i(t)\frac{w_i(t)}{2}$ $= w_i(t) + \frac{1-q_i(t)}{\tau_i} - \frac{1}{2}q_i(t)x_i^2(t)\tau_i$
	Link control	$\dot{b}_l = \begin{cases} y_l(t) - c_l & if\ b_l(t) > 0 \\ [y_l(t) - c_l]^+ & if\ b_l(t) = 0 \end{cases}$ $\dot{r} = -\alpha_l c_l (r_l(t) - b_l(t))$ $p_l = m_l(r_l)$
	Utility function	$U_i(x_i) = \frac{\sqrt{2}}{\tau_i}\tan^{-1}\left(\frac{\tau_i x_i}{\sqrt{2}}\right)$
Vegas (FIFO)	Source sending rate control	$w_i(t+1) = \begin{cases} w_i(t) + \frac{1}{D_i(t)} & if\ \frac{w_i(t)}{d_i} - \frac{w_i(t)}{D_i(t)} < \alpha_i \\ w_i(t) - \frac{1}{D_i(t)} & if\ \frac{w_i(t)}{d_i} - \frac{w_i(t)}{D_i(t)} > \alpha_i \\ w_i(t) & else \end{cases}$
	Link control	$\dot{p}_l = \begin{cases} \frac{1}{c_l}(y_l(t) - c_l) & if\ p_l(t) > 0 \\ \frac{1}{c_l}[y_l(t) - c_l]^+ & if\ p_l(t) = 0 \end{cases}$
	Utility function	$U_i(x_i) = \alpha_i d_i \log x_i$

In the Reno model, τ_i is the round-trip time (RTT) of source *i*, $b_l(t)$ and $r_l(t)$ denotes the instantaneous and average queue length of link *l* at time *t* respectively, which has the link capacity c_l. The packet dropping or marking probability on link *l* is determined by the RED function $m_l(r_l(t))$.

In the Vegas model, d_i and $D_i(t)$ denotes the round-trip propagation delay and round-trip time (propagation plus queuing delay, etc.) of source *i*, respectively. α_i is a system parameter, which has a significant impact on fairness analysis.

A source utility function, taking the form of integral of inverse function source aggregate price q of source sending rate x, is introduced for the purpose of performing an optimization interpretation for the equilibrium. In other words, the equilibrium rate is intended to solve the following maximization problem of individual source's profit:

$$\underset{x_i}{Maximize}\left(U_i(x_i) - x_i q_i\right)$$

To align the individual optima with the global optima (maximization of the aggregate utility across all sources with the link capacity constraints), a duality approach is introduced in [LL99], where the link prices come in play as Lagrange multipliers.

[Kelly99] uses the system of differential equations about utility and rate functions to describe the network traffic. A Lyapunov function is given for the system of differential equations under different assumptions of the weight parameter to show that the zero solution (i.e. utility maximization) is stable. This analytical method is applied to the congestion avoidance part of the TCP algorithm and the fairness between rate control algorithms are discussed as well.

Utility function in the above discussion is assumed to be an increasing, strictly differentiable concave function of source sending rate with a positive and decreasing derivative. However, this assumption is very unlikely to maintain its validity in a real network system. As a matter of fact, none of the relations between any network control and performance parameters exhibits strict smooth (or differentiable) property. This is the main reason why we want to explore and lead stochastic approximation methods into the network analysis and end-to-end transport control. Furthermore, in the above mathematical models, it is not obvious to relate the economic interpretation of the utility function to a specific performance metrics parameter in a network context.

2.3 Overlay Networks

Unfortunately, Internet has not gained much improvement since mid 90's regardless of all the efforts made by the scientific community. The DropTail queuing management, which simply drops an incoming packet to a full data link buffer, still dominates the current router market.

It is well known that the cooperation of the routers is necessary to achieve the optimal solution because routers have the information about the state of network congestion. Hence, a great deal of effort has been put into designing and developing algorithms that take advantage of this amount of information provided by routers. However, the Internet implementation of such algorithms requires changes on various network components in the widely distributed nodes, which are managed and maintained by different organizations all over the world. Given the enormous size of the Internet, any change of the existing network infrastructures is very difficult to make in a short time so that most of the recent theoretical research proposals are not implemented and are not of much use in practice. This motivates us to explore ways to deploy new protocols in overlay network as discussed below.

According to TCP/IP and OSI reference models, the protocol stack up to transport layer is normally provided as part of the Operating System kernel and is responsible for communications with other nodes [Stevens98]. The user applications handle all the details of data processing and talk to the lower protocol layers through an Application Programming Interface (API).

Overlay network is a virtual layer built above the infrastructure network and each of its edges corresponds to a unicast path between two end systems in the underlying network [Rao01, Hypercast, ABKM01]. As illustrated in Figure 2.10, the bottom base network is used to support the upper logical overlay network.

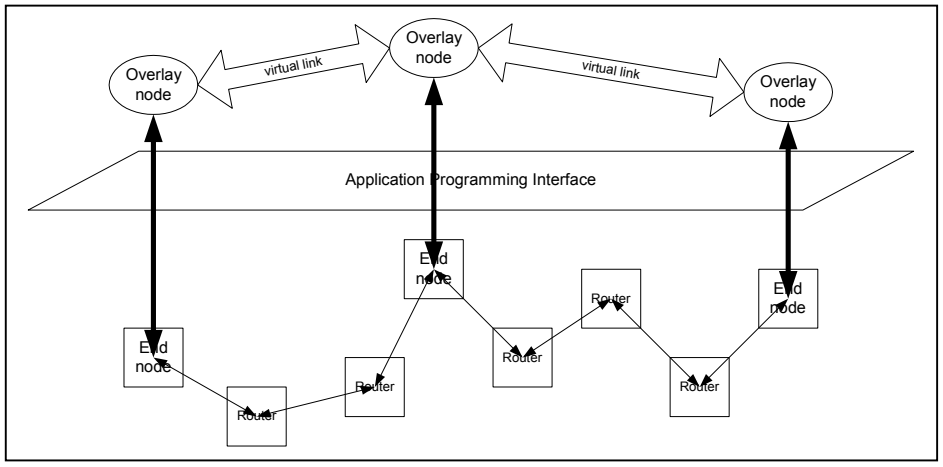


Figure 2.10 Overlay network

Since the overlay network resides at the application level, its deployment requires no changes to the existing Internet infrastructure. This is the most important feature of an overlay network, and is also the primary reason for us to implement our transport protocols in the overlay network. Besides, the design of overlay network is flexible with the requirement to applications, and the increased control and the adaptable nature make the overlay network more robust. Overlay networks have been used in many applications to deploy or test new protocols and services with minimal affecting the lower IP infrastructure [JGJKO00, ABKM01, TH98].

The NetLets idea was first introduced by Rao in [Rao01] to minimize end-to-end message delays using overlay routers, which are daemons suitably deployed over the wide area network. These daemons communicate with each other to form an overlay network [RRCWI01].

CHAPTER 3 NETWORK TRAFFIC MEASUREMENT AND ANALYSIS

The conventional flow control mechanism implemented in TCP is through a congestion window. The sending TCP waits a period of RTT for acknowledgements and decides whether to enlarge or reduce the congestion window accordingly. Therefore, the TCP sending rate is approximately determined by the ratio of congestion window size and RTT. One alternative way to provide flow control is that the sender sends only one data packet at a time and wait a certain period that is much shorter than RTT to spread the data packets evenly.

In this chapter, we design a network transport control model based on the current Internet implementations of both UDP and TCP. This model combines the above two flow control mechanisms and simultaneously maintains two parameters, congestion window and sleep time to control the source transmission rate. The TCP flow control is a special case of our model where the sleep time is set as RTT, while the alternative way of flow control is another special case of our model where the congestion window is fixed at one packet.

The sleep time is interchangeably referred to as idle time, waiting time, or inter-cwin delay within the context of this dissertation. No mechanism of end-to-end congestion control is employed at the moment. The network traffics based on this transport control model are measured from extensive experiments over a long time span and analyzed using statistical methods to determine the main effects of these two factors and the interaction effects between them.

3.1 Network Transport Control Model

We design a window-based network transport control model using both UDP and TCP to study the characteristics of packet flow over network and take effective measure of network traffic. The testing data is generated by the sender (or, equivalently, client or source node) and then actively delivered to the receiver (or, equivalently, server or destination node) via UDP, taking advantage of UDP's transmission maneuverability. The sender informs the receiver about the initialization and termination of the testing data delivery process via TCP, taking advantage of TCP's transmission reliability. The transport mechanism with sending rate control of UDP datagrams and transmission process handling through TCP connection is shown in Figure 3.1.

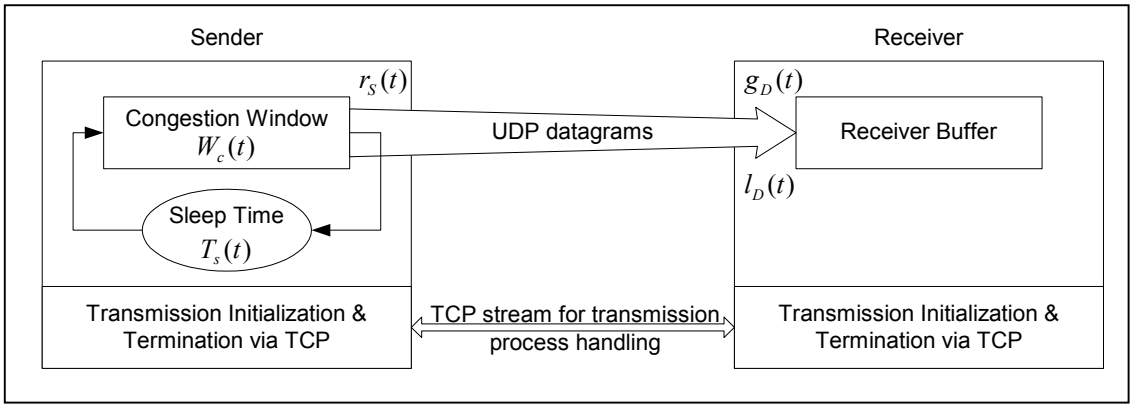


Figure 3.1 Sending rate control mechanism for network traffic measurements

This simple window-based flow control scheme maintains two parameters, congestion window and sleep time [RWIM02], both of which are intended to control the source sending rate, expectedly with different effects on the network performance.

Congestion Window: denoted by $W_c(t)$, is a counterpart of *cwnd* in TCP. It represents the number of UDP datagrams that can be sent continuously as fast as the computer and communication hardware resources (CPU, memory, NIC speed, channel bandwidth, etc.) allow.

Sleep Time: denoted by $T_s(t)$, is also referred to as idle time, waiting time, or inter-cwin delay. It represents the amount of period the sender suspends right after sending a full congestion window of UDP datagrams until starting to send the very next full congestion window of UDP datagrams.

There are three main steps involved in the network traffic measurements using the window-based flow control model.

Step 1. **Initialization**: The sender creates a certain amount of testing data, typically 2M, 5M, or 10M bytes, and initializes the transmission by sending the testing data information such as message length, UDP datagram size, user-defined header format etc. to the receiver through a pre-connected TCP channel. Both sender and receiver set on their own starting timer for later reference.

Step 2. **Transmission**: The sender divides the testing data into parts of MTU size minus both UDP and IP header lengths (i.e. MTU – 8 bytes – 20 bytes), assigns each part a continuous sequence number, and sends them to the receiver sequentially as a set of UDP datagrams at a fixed rate determined by the congestion window and sleep time. The receiver reads the datagrams in the order they arrive. Neither are received datagrams acknowledged, nor are lost datagrams retransmitted.

Step 3. **Termination**: The sender notifies the receiver of the end of the transmission process by sending an ending signal to the receiver through TCP stream as soon as all UDP datagrams are sent. Upon receiving the ending signal, the receiver at the destination node turns off timer and calculates average goodput $g_D(t)$ as well as loss rate $l_D(t)$. Any datagrams arriving after the ending signal are received and claimed as late datagrams.

Based on the above flow control model, the instantaneous source rate $r_S(t)$ can be directly calculated as:

$$r_S(t) = \frac{W_c(t)}{T_s(t) + T_c(t)} = \frac{W_c(t)}{T_s(t) + \frac{W_c(t)}{BW}} = \frac{1.0}{\frac{T_s(t)}{W_c(t)} + \frac{1.0}{BW}} \tag{3.1}$$

where $T_c(t) = \frac{W_c(t)}{BW}$ is the time spent on continuously sending out a full congestion window of UDP datagrams, which is determined by the congestion window size and communication hardware resources, mostly the system bandwidth *BW*, i.e. the maximum speed at which the sender host can generate the signal and put it on wire. The sleep time $T_s(t)$ is usually comparable to the round trip time (RTT), hence generally we have the relationship $T_c(t) < T_s(t)$ due to the long delay, widely available high bandwidth, and

relatively small packet size in the wide-area networks. Note that the inter-cwin delay in TCP flow control is always fixed at the value of RTT.

According to Equation (3.1), we may control the source-sending rate $r_S(t)$ by adjusting either congestion window $W_c(t)$ or sleep time $T_s(t)$ respectively, or both simultaneously. Seeking for the dynamic combinations of $W_c(t)$ and $T_s(t)$ to achieve the optimal sending rates satisfying different performance requirements is the main objective of our research to be conducted in this dissertation.

3.2 Measurement Data Setup

The window-based flow control mechanism is implemented and installed on the host at LSU, resource.rrl.lsu.edu as a server (or receiver), and the host at ORNL, ozy4.csm.ornl.gov as a client (or sender). The path shown in Figure 1.1 provided by *traceroute* shows that before the network path rerouting, ORNL is connected to ESnet, which peers with Abilene network in New York. Abilene runs from New York via Washington DC and Atlanta to Houston, where it connects to LSU via a regional network. In terms of network distance, these two sites are separated by more than two thousand miles, and both ESnet and Abilene have significant network traffic.

The client at ORNL generates a message of a certain size and sends it to the server at LSU as a set of UDP datagrams at a fixed rate at a time for multiple times with different sending rates. We maintain a constant sending rate of UDP datagrams during each run of the message transmission by fixing both congestion window and idle time in the flow control mechanism. To illustrate how the different sending rates affect the network transport performance, we list in Table 3.1 the performance measurements from two runs of the message transmission of 2M bytes, one run in fast sending mode with congestion window of 100 datagrams and one run in slow sending mode with congestion window of 10 datagrams. Both runs have the same sleep time of 100 milliseconds and UDP datagram size of 1472 bytes (1500 – 8 – 20). Note that the resolution of the system defined function *nanosleep()* is about 1 millisecond or 10 milliseconds depending on different systems. We designed and implemented our version of sleep function *myusleep()* that has a resolution as precise as one microsecond.

Table 3.1 Performance measurements from two runs of message transmission in two different sending modes

Sending mode	Sender (Source)				Receiver (Destination)		
	cwnd (dgs)	Idle time (ms)	No. of dgs sent	Sending rate (Mbps)	No. of dgs recvd	Goodput (Mbps)	Loss rate (%)
Fast	100	100	1392	7.6878	723	3.5114	48.0344
Slow	10	100	1392	1.0715	1390	1.0707	0.0938

From Table 3.1, we notice that the high source sending rate results in a high UDP datagram loss rate, which drastically reduces the effective receiver goodput, i.e. the datagram receiving rate at the destination node in the fast sending mode, while the slow sending mode exhibits a quite stable goodput with very few datagram losses.

Plot (a) and (b) in the upper part of Figure 3.2 show curves of datagram sending and receiving time, represented by the *x* axis, versus datagram sequence numbers, represented by the *y* axis, based on the measurements from the above two runs of message transmission. For illustrative purposes, the ending signal transferred through TCP stream is particularly labeled as the maximum UDP datagram sequence number plus 1. The end phase of the message transmission in the slow sending mode and the start phase of the message transmission in the fast sending mode are enlarged in Plot (c) and (d) in the lower part of Figure 3.2, respectively for a closer view.

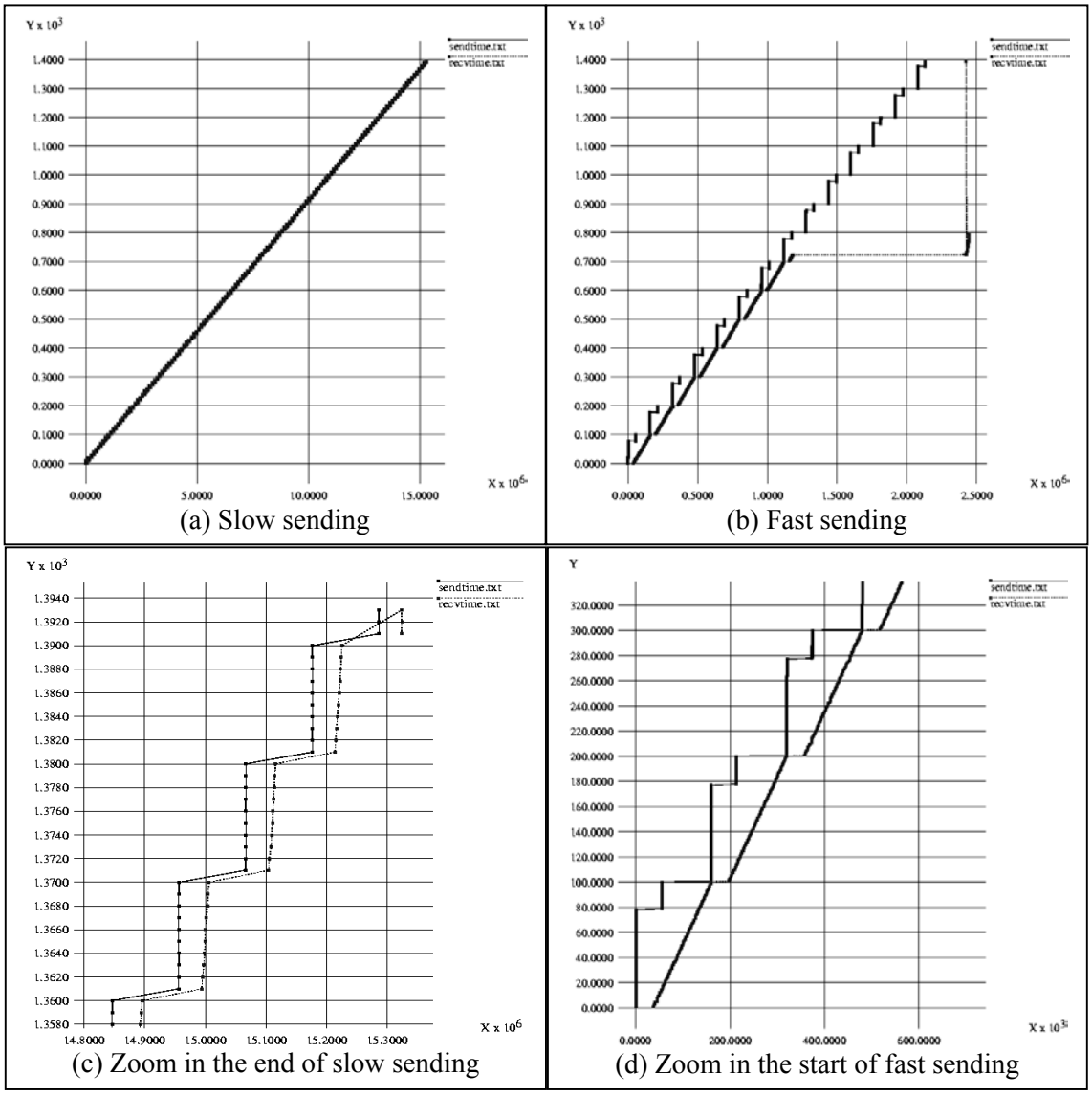


Figure 3.2 Sending time versus sequence numbers of sent datagrams

One interesting thing observed in Figure 3.2 is that late datagrams may arrive after the TCP ending signal in spite that all UDP datagrams are sent before the TCP ending signal. For example, the last two datagrams in Plot (c) of Figure 3.2 and the set of late datagrams with sequence numbers ranging from 724 to 797 in Plot (b) are received after the completion of UDP datagram transmission, which is indicated by the receipt of the TCP ending signal that arrived earlier. This phenomenon is possibly due to the discriminating

IP packet flow regulation for UDP datagrams and TCP segments implemented in routers or hosts. In other words, some UDP datagrams, which are claimed lost by the receiver at the application layer, may be delayed and held up at some intermediate routers or end hosts for an indefinite time in times of network congestion. This observation is consistent with the delay-throughput relation described earlier in Chapter 1: the more heavily the network is loaded, the longer the packets are delayed.

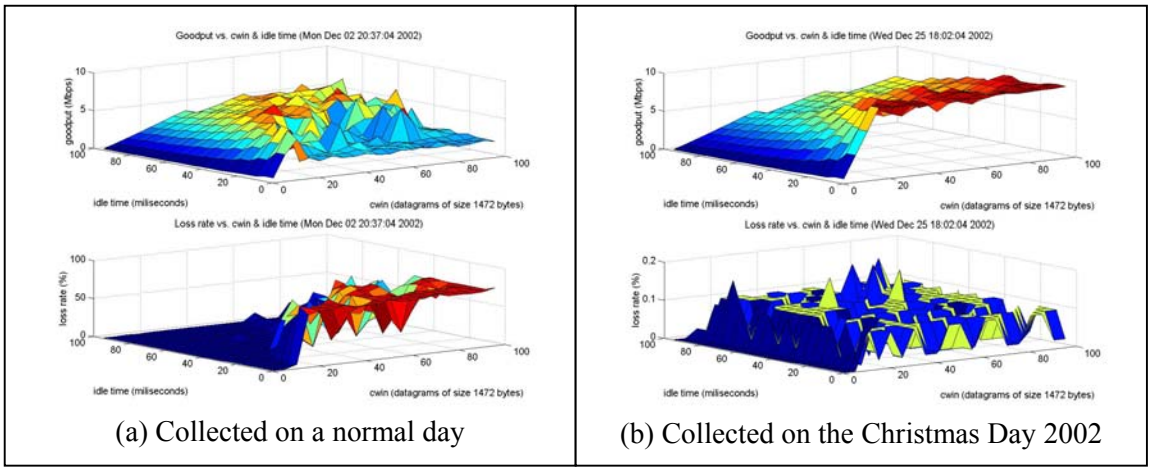
The file transfer protocol named Tsunami takes advantage of this feature of data transmission through network to achieve high individual throughput [Tsunami]. However, its extreme aggressiveness grabs most of the bandwidth disregarding the network congestion condition and hence causes serious fairness problems to other concurrently running network applications based on conformant TCP. Experimental details about Tsunami performance are given later in Chapter 5, where we also modify the current transport control model to implement a Tsunami-like protocol for the purpose of comparison. Nevertheless, in many practical network applications, it is always reasonable and necessary to assume that an extremely delayed datagram is lost as far as the satisfactory real-time network performance is concerned.

Another thing worthy of being pointed out is that the plots in Figure 3.2 are sort of misleading. It appears in the figure that the full congestion window of 10 datagrams during the slow message transmission are sent almost at the same time, and the full congestion window of 100 datagrams during the fast message transmission has been "divided" into two parts, each of which are sent individually almost at the same time. This anomalous observation could be explained by the fact that UDP does not require a connection so that the UDP socket is always writable as long as the send low-water mark for a UDP socket is less than the available send buffer size, which is the default relationship [Stevens98]. The UDP I/O function *sendto()* will immediately return after placing the datagram (or the packet through IP layer) in the outgoing link queue if the send buffer is not fully filled as is the case of slow sending; otherwise it will wait until enough space in the send buffer is released and the UDP socket becomes writable again as is the case of fast sending. The written data packets remain in the FIFO outgoing queue until the system scheduler activates the signaling process, which continuously generates bit signals for the packets (or the frames in the data link layer) in the queue and wires them through the physical medium at the maximum rate of the system bandwidth. Therefore, the real datagram sending time is actually in turn delayed for each succeeding datagram in the same congestion window, and the idle time between two adjacent congestion windows observed in Figure 3.2 accounts for the sum of the preset sleep time $T_s(t)$ and the transmission time $T_c(t)$ of the (full or partial) congestion window of UDP datagrams. This explanation is justified by the regularly delayed receiving time of the datagrams in the same congestion window measured at the destination node.

As mentioned in Step 2 of the network traffic measuring process, no congestion control mechanism is implemented in the transport control model at the moment because these experiments are solely for the purpose of network traffic analysis. Specifically speaking, neither does the server acknowledge any datagrams that are successfully received nor the client detects network congestion and retransmits lost datagrams. In order for the network to discard any drifting packets that are extremely delayed, the sender always waits a long enough period of time at the end of each run so that the current run of the message transmission has no impact on the succeeding one.

With regard to the transmission rate at the source node, although it is analytically determined by Equation (3.1), we compute the average source rate $r_S(t)$ online as the number of sent UDP datagrams divided by the transmission duration. On the other side at the destination node, the average goodput $g_D(t)$ and loss rate $l_D(t)$ are measured as the number of successfully arriving UDP datagrams and the number of lost UDP datagrams respectively, both of which are divided by the time elapsed since the first datagram is delivered.

The various source transmission rates are achieved through the independent adjustment made on either congestion window or idle time. Particularly, we conduct the transport control experiment by varying the congestion window from 1 to 100 at a step of 5 UDP datagrams and by varying the sleep time from 1 to 100 at a step of 5 milliseconds independently. The network traffic measurements are collected between the client host ozy4.csm.ornl.gov at ORNL and the server host resource.rrl.lsu.edu at LSU. The whole set of experiment, i.e. multiple runs of the message transmission at various fixed sending rates, is repeated over hours, days, weeks, and months to extensively measure network traffic behaviors. For the purpose of comparison, we plot in Figure 3.3 the network traffic that is measured on a normal day and the Christmas Day in 2002 respectively.



(a) Collected on a normal day

(b) Collected on the Christmas Day 2002

Figure 3.3 Goodput and loss rate response surface plot with two random-effect factors: congestion window, idle time from Internet traffic measurements

The top plots in Figure 3.3 represent the average goodput $g_D(t)$ and the bottom plots represent the loss rate $l_D(t)$ at the destination node. Each point in the horizontal plane corresponds to a pair of congestion window and idle (i.e. sleep) time, which determines the source transmission rate $r_S(t)$ as defined by Equation (3.1). From our measurements over 6 months, we found that except for several special holidays of the year such as the Christmas Day as shown in Plot (b) of Figure 3.3, the network traffic exhibits a very similar two-phase pattern in most of time as shown in Plot (a).

In the first phase, there is a trend of monotonic increase in goodput $g_D(t)$ as sending rate $r_S(t)$ is increased while the loss rate $l_D(t)$ remains at an extreme low level. After the sending rate reaches a certain transition point, the system enters the second phase where the goodput $g_D(t)$ starts suffering irregular decrease due to congestion collapse indicated by the high datagram loss rate $l_D(t)$. This overall behavior is quite stable although the

transition position dividing these two phases and the goodput shape may slightly vary over time in the presence of diverse background traffic. We also observed that the plots of goodput and loss rate are quite non-smooth because of dynamically changing network condition that produces the randomness involved in packet delays and losses.

As for the traffic measurements in Plot (b) of Figure 3.3, the goodput exhibits a trend of persistent increase with a low loss rate when the sending rate continuously increases up to the system bandwidth. This exceptional phenomenon could be attributed to the fact that the transmission bottleneck in the Internet is not on the backbone, but usually on the end-systems. On such a special day as the Christmas Day, these two end-systems, ORNL and LSU, have very few users and the local area networks are almost empty. Therefore, the experiment conducted during this particular period of time is allowed to use the hardware resources to achieve the goodput up to the system bandwidth 10Mbps.

To further illustrate the effects of different sending rates on the network performance, we plot in Figure 3.4 and Figure 3.5 the destination goodput and loss rate, respectively, as functions of the sending rate based on the measurements that are collected on normal days between two hosts at ORNL and LSU. These plots can be easily generated from Figure 3.3 by fixing the idle time and increasing the congestion window, which corresponds to taking vertical slices of the three-dimensional plots parallel to the congestion-window axis.

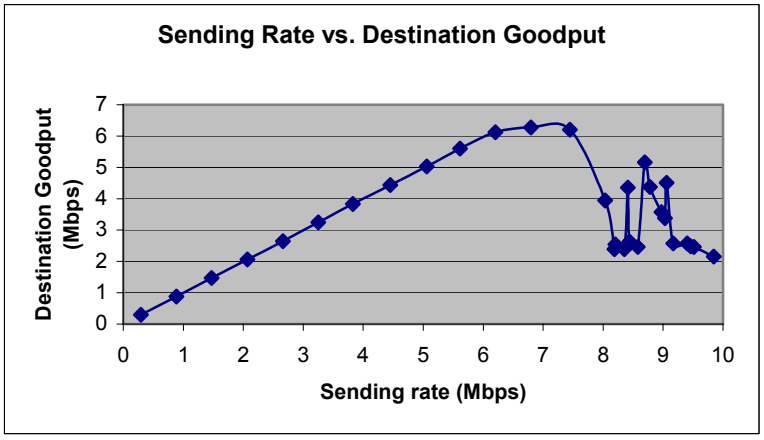


Figure 3.4 Source sending rate vs. destination goodput

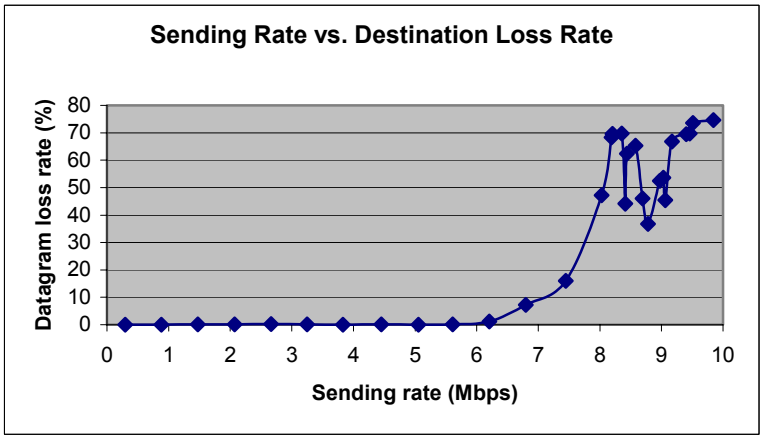


Figure 3.5 Source sending rate vs. destination loss rate

The two-phase pattern of the network performance from the experiments conducted on normal days is clearly shown in Figure 3.4 and Figure 3.5. This type of overall goodput response is well known and has been modeled as fixed, strictly increasing, and continuously differentiable concave utility functions of source sending rate in a number of recent theoretical works on transport control [LPD02, Kelly99]. However, in practical applications, such ideal characterization is not feasible in wide-area networks. When source rate $r_S(t)$ is fixed at r, the goodput $g_D(t)$ is a random variable, which is jointly distributed with distribution $G(g_D(t), r)$.

By running the experiment with the same control parameters for a sufficient large number of times and computing the mean values of the destination goodput $g_D(t)$, we obtain the expected value of the destination goodput corresponding to the fixed sending rate r:

$$M_D(r) = E[g_D(t) \mid r_S(t) = r] = \int g_D G(dg_D, r) \tag{3.2}$$

where $G(\cdot, r)$ is an unknown distribution function of real-valued random variable g_D at a given constant sending rate r. We refer to $M_D(r)$ as the *destination goodput response regression*. The long-time-span Internet measurements show that $M_D(r)$ is considerably stable and experiences very slight variation in the presence of either on-host or off-host background traffic during normal days except for the extreme behavior observed on several special days of the year. Nevertheless, we shall consider a dynamically changing response regression in TCOU as presented in detail in Chapter 4 and Chapter 5.

3.3 Statistical Analysis of Network Traffic

As discussed above, the window-based flow control mechanism has two different ways to change the source transmission rate. Now we are interested in understanding how the various transmission rates that are controlled by different combinations of congestion window and idle time affect the network transport performance. In order to explore the network performance pattern and the stochastic nature of network traffic, a series of extensive Internet experiments have been conducted over a time span longer than six months. The collected measurements are used as the data sets for the experimental statistics methods described below.

3.3.1 Introduction to Experimental Statistics

Many systems in the physical world can be considered as a function of the relations that persist between their parts, each of which is referenced by either a *condition variable* or a *response variable.* Some simple systems have deterministic and straightforward relations between variables that could even be precisely expressed in mathematical forms. However, in some complex systems like the network transport control system under research, the nature of inter-variable relations is not obvious and the only available information about the system is the *measurements* made on the *observational unit* after carefully and methodically applying *treatments* to the *experimental unit*. A data set usually consists of levels of treatments and corresponding measurements. The purpose of using statistics is to obtain meaningful information from data sets.

There are many different techniques used in experimental statistics, such as regressions (single and multiple), factorial experiments (two-way, three-way, etc.), and experimental designs (*Completely Randomized Design* -- CRD, i.e. one-way ANOVA, *Randomized Complete Block Design* -- RCBD, *Latin Square*, *Split-Plot*, etc.), and so on. Regression is

a well-known method that is used to investigate and quantify the relationships between variable quantities. Factorial experiment has each combination of all factor levels applied to experimental units and employs the ANOVA (analysis of variance) methodology for examining the effect of factors on the same type of unit [FW97]. Experimental design uses "blocking" procedure to subdivide the experimental units into groups before assigning them to different factor levels with the intention of reducing the estimate of variance used for inference. Choosing an appropriate approach for a specific problem has been one of the main tasks in statistics.

3.3.2 The Two-Factor Factorial Experiment

In our transport control model, two factors, congestion window and sleep time, are simultaneously applied to each experimental unit, the datagram transmission at the source node. We are concerned with the analysis of data corresponding to two-factor experiments, in which each combination of factor levels is considered. Particularly, we are interested in the *main effects*, i.e. the differences in the mean goodput response across the levels of either congestion window or sleep time when viewed individually, and *interaction effects*, i.e. the differences of the main effect goodput responses for one factor across levels of the other. The two-way ANOVA turns out to be a suitable approach for satisfying our research purpose.

Besides the decision on statistical methods, it is also very important to determine whether a *fixed*, *random*, or *mixed effects model* is used before we conduct the two-factor factorial experiment because the ANOVA analysis results considerably depend on the model we select for the method. If the subjects are a random sample of a population, the corresponding control variable is known as a random effect. That is to say, if we repeat the study, we could have a different sample of subjects. In contrast, the variable that always has the same values or levels (pre and post) in any repeat of the study is a fixed effect. For example, sex has a deterministic value, male or female in every human being sample. A model with both fixed and random effect treatments is called a mixed model. Apparently, in our data transmission experiments, neither congestion window nor sleep time has constant levels. The specific values we choose for congestion window from 1 to 100 with an interval of 5 datagrams and for sleep time from 1 to 100 with an interval of 5 milliseconds are only a small portion of the population, which consists of any valid combinations of congestion window and sleep time. Therefore, both factors are considered as random effect treatments.

3.3.3 General Linear Model, Assumptions, and ANOVA Table

In experimental statistics, observations are always conveniently expressed in terms of a *statistical model*, which describes the behavior of the observed values in response to one or more of the control parameters of the sampled distribution. The *analysis of variance model* is based on partitioning *sums of squares* (SS, or SOS) and using ratios of the resulting mean squares as test statistics for making inferences on parameters that represent the various means. The *regression model* relates a single response variable to other independent variables and makes predictions about the former. These two models have different analysis procedures and hence are used in different applications. As discussed in later chapter, a linear regression model is used in the measurement based path bandwidth estimate by a NetLet daemon. However, in some situations the aspects of both models may be used. Two typical examples are *contrasts* and *orthogonal*

polynomials, both of which are used in analysis of variance situations and are actually regression analysis using specially coded independent variables [FW97]. The unified treatment of regression and analysis of variance is referred to as the *general linear model*. To facilitate understanding of the network transport control analysis, we use a linear statistical model for the two-random-effects factorial experiment of congestion window and sleep time as follows:

$$g_{ijk} = \mu + c_i + s_j + (cs)_{ij} + \varepsilon_{k(ij)} \tag{3.3}$$

where

g_{ijk} : k-th observed value, k = 1, 2, …, n of the destination goodput response variable $g_D(t)$ under the combinatorial treatment defined by the i-th level of congestion window and j-th level of sleep time.

μ : reference value, usually referred to as grand or overall mean, calculated as the sum of all observation values divided by the total number of observations.

$c_i, i = 1, 2, \cdots, a$: main effects of congestion window on goodput response, calculated as the difference between the mean response of the subpopulation comprising the i-th level of congestion window and the grand mean μ.

$s_j, j = 1, 2, \cdots, b$: main effects of sleep time on goodput response, calculated as the difference between the mean response of the subpopulation comprising the j-th level of sleep time and the grand mean μ.

$(cs)_{ij}$: interaction effects between congestion window and sleep time, calculated as the difference between the mean goodput response in the subpopulation defined by the combination of the factor levels of c_i and s_j, and the mean goodput response when there only exist main effects of either c_i or s_j.

$\varepsilon_{k(ij)}$: random error. The parentheses around subscript variable i and j can be considered as a cell defined by i-th level of congestion window and j-th level of sleep time. The random error $\varepsilon_{k(ij)}$ represents the variation among n goodput observations nested in the cell (i, j), and is calculated as the difference between k-th observation g_{ijk} nested in cell (i, j) and the cell mean $\bar{g}_{(ij)}$.

In support of the linear model defined in Equation (3.3), we have the following assumptions for the variables described above.

(1) The grand mean μ is a constant.

(2) All random variable c_i, s_j, $(cs)_{ij}$, and $\varepsilon_{k(ij)}$ are independent.

(3) Main effects $c_i, i = 1, 2, \cdots, a$ are identical and independent variables, with a zero-mean normal distribution, or equivalently, $c_i, i = 1, 2, \cdots, a$ iid $\sim N(0, \sigma_c^2)$.

(4) Main effects $s_j, j = 1, 2, \cdots, b$ are identical and independent variables, with a zero-mean normal distribution, or equivalently, $s_j, j = 1, 2, \cdots, b$ iid $\sim N(0, \sigma_s^2)$.

(5) Interaction effects $(cs)_{ij}$ are identical and independent variables, with a zero-mean normal distribution, or equivalently, $(cs)_{ij}$ iid $\sim N(0, \sigma_{cs}^2)$.

(6) Random errors $\varepsilon_{k(ij)}$ have a mean of 0 and a common variance equal to σ_ε, or equivalently, $\varepsilon_{k(ij)} \sim N(0, \sigma_\varepsilon^2)$.

According to the linear statistical model and assumptions, we tabulate the standard two-way ANOVA table for random-effects model in Table 3.2, where by convention, the congestion window is represented by Factor A and the sleep time is represented by Factor B. There are $n = 5$ observations made in one cell defined by a pair of congestion window and sleep time. The congestion window and sleep time have the same number of levels $a = b = 20$.

Table 3.2 ANOVA table for two-way random effect factorial experiment

Source	Sum of squares (SS)	Degree of freedom (df)	Mean squares (MS)	Expected Mean Square (EMS)
Factor A (cwnd)	SSA	$a - 1$	MSA	$\sigma_\varepsilon^2 + n\sigma_{cs}^2 + bn\sigma_c^2$
Factor B (sleep)	SSB	$b - 1$	MSB	$\sigma_\varepsilon^2 + n\sigma_{cs}^2 + an\sigma_s^2$
Interaction AB	SSAB	$(a - 1)(b - 1)$	MSAB	$\sigma_\varepsilon^2 + n\sigma_{cs}^2$
Error	SSE	$ab(n - 1)$	MSE	σ_ε^2
Totals	TSS	$abn - 1$		

Since we treat both factors as random effects, the quantity MSAB will be used as the error term to test the main effects as shown later in the output produced by the SAS program. Taking variance of Equation (3.3) results in the following:

$$\begin{aligned} Var(g_{ijk}) &= Var[\mu + c_i + s_j + (cs)_{ij} + \varepsilon_{k(ij)}] \\ &= \sigma_c^2 + \sigma_s^2 + \sigma_{cs}^2 + \sigma_\varepsilon^2 \end{aligned} \tag{3.4}$$

From Equation (3.4) we know that there are four variance components in the model we selected, which are respectively estimated as follows:

$$\hat{\sigma}_\varepsilon^2 = MSE,$$

$$\hat{\sigma}_{cs}^2 = \frac{MSAB - MSE}{n},$$

$$\hat{\sigma}_c^2 = \frac{MSA - MSAB}{bn} = \frac{MSA - MSE - n\hat{\sigma}_{cs}^2}{bn},$$

$$\hat{\sigma}_s^2 = \frac{MSB - MSAB}{an} = \frac{MSB - MSE - n\hat{\sigma}_{cs}^2}{an}.$$

3.3.4 Tests of Hypotheses

Statistical analyses in practice are always carried out by computer software. Here we will use SAS, the most common large-scale data analysis software package to perform the

ANOVA analysis of the random-effects two-way factorial experiment. SAS is the abbreviation of **S**tatistical **A**nalysis **S**oftware. It is a major statistical software package developed by the SAS Institute Incorporation [SAS]. The main purpose of the SAS software package is to read in, process, and output statistical information from data sets. The results of two-way analysis using a GLM procedure for the network traffic measurements are compiled and listed in Table 3.3. Please refer to the Appendix for the details about the SAS program.

Table 3.3 Compiled outputs from SAS program

Sleep time & congestion window vs. Goodput
The GLM Procedure

Class Level Information

Class	Levels	Values
sleeptime	20	1 11 16 21 26 31 36 41 46 51 56 6 61 66 71 76 81 86 91 96
cwin	20	1 11 16 21 26 31 36 41 46 51 56 6 61 66 71 76 81 86 91 96

Number of observations 2000

Tests of Hypotheses for Random Model Analysis of Variance

Dependent Variable: goodput

Source	DF	Type III SS	Mean Square	F Value	Pr > F
sleeptime	19	565.263754	29.750724	5.26	<.0001
cwin	19	1923.400159	101.231587	17.91	<.0001
Error	361	2040.027317	5.651045		

Error: MS(sleeptime*cwin)

Source	DF	Type III SS	Mean Square	F Value	Pr > F
sleeptime*cwin	361	2040.027317	5.651045	10.35	<.0001
Error: MS(Error)	1600	873.450238	0.545906		

Tests for Normality

Test	--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.875659	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.180962	Pr > D	<0.0100
Cramer-von Mises	W-Sq	22.33985	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	105.8061	Pr > A-Sq	<0.0050

The main effects of congestion window and sleep time and the interaction effects between them as well as the normal distribution of the residuals under the linear model are of our main concerns. Therefore, we have the following hypotheses of interests and the test of each hypothesis is given accordingly.

(1) Hypothesis of main effects of congestion window (Factor A) on goodput response

$$H0:\sigma_c^2 = 0 \quad H0:\sigma_A^2 = 0$$
or
$$H1:\sigma_c^2 \neq 0 \quad H1:\sigma_A^2 \neq 0$$

As shown in Table 3.3, the mean square of the interaction effects is used as the error term for statistic test of main effects in random effects model. Therefore, the F value is computed as:

$$F^* = MSA/MSAB = 101.231587/5.651045 = 17.91$$

with numerator degree of freedom *ndf* equal to 19 and denominator degree of freedom *ddf* equal to 361. The corresponding *p-value* is less than 0.0001, based on which we reject

H0. In other words, the observation data strongly indicates that the congestion window has a significant effect on the destination goodput.

(2) Hypothesis of main effects of sleep time (Factor B) on goodput response

$$H0:\sigma_s^2 = 0 \quad H0:\sigma_B^2 = 0$$
$$\text{or}$$
$$H1:\sigma_s^2 \neq 0 \quad H1:\sigma_B^2 \neq 0$$

Same as above, the mean square of the interaction effects is used as the error term for statistic test of main effects, and the *F* value is computed as:

$$F^* = MSB / MSAB = 29.750724/5.651045 = 5.26$$

with numerator degree of freedom *ndf* equal to 19 and denominator degree of freedom *ddf* equal to 361. The corresponding *p-value* is less than 0.0001, based on which we reject *H0*. In other words, the observation data strongly indicates that the sleep time also has a significant effect on the destination goodput.

(3) Hypothesis of interaction effects between congestion window and sleep time (Factor A and Factor B) on goodput response

$$H0:\sigma_{cs}^2 = 0 \quad H0:\sigma_{AB}^2 = 0$$
$$\text{or}$$
$$H1:\sigma_{cs}^2 \neq 0 \quad H1:\sigma_{AB}^2 \neq 0$$

As shown in Table 3.3, the mean square of error MSE is used as the error term for statistic test of interaction effects. Therefore, the *F* value is computed as:

$$F^* = MSAB / MSE = 5.651045/0.545906 = 10.35$$

with numerator degree of freedom *ndf* equal to 361 and denominator degree of freedom *ddf* equal to 1600. The corresponding *p-value* is less than 0.0001, based on which we reject *H0*. In other words, the observation data strongly indicated that there is a significant interaction between the congestion window and sleep time.

(4) Hypothesis of residual normality under linear model

Shapiro-Wilk method is used to verify the residual normality assumption. As shown in Table 3.3, the SAS program calculates the statistic *W* equal to 0.875659, and the corresponding *p-value* ($Pr < W$) is less than 0.0001, based on which, we infer that our residual normality assumption is valid in the light of data.

As a conclusion of this section, both congestion window and sleep time turn out to have significant effects on the destination goodput response according to the two-way ANOVA analysis we performed on the network traffic measurements collected from the transport control model. We also learned that these two control parameters are not independent but have a strong interaction between them. We will consider both of them in our later study of the transport control methodology based on stochastic approximation methods.

CHAPTER 4 TCOU FOR GOODPUT STABILIZATION

4.1 Introduction

There exist a number of network applications serving a wide variety of purposes, which pose very different throughput requirements. A transport control mechanism that aims to equally allocate network resources or unfairly maximize individual throughputs is apparently not the most efficient way of utilizing the bandwidth from an overall perspective.

In some cases where a certain level of goodput sufficiently guarantees the quality of service provided by an application, any extra bandwidth allocated to that application beyond its threshold could be seen as a waste of resources if the coexistent network applications are suffering from the lack of bandwidth. On the other hand, the goodput achieved by some network application must remain above a certain level to achieve satisfactory performance even if its current share of the bandwidth has been used up. Most of the real-time remote controls over wide area networks and multimedia data streams such as coordinated visualization of distributed datasets as well as remote medical diagnosis serve as good examples of such applications. The fact of abundance of these "premium services" motivates us to explore a way of "allocating resources as needed", instead of "allocating resources equally" as is the case in TCP.

The current implementation of TCP treats all participating TCP sessions equally and the bandwidth is shared among them through AIMD congestion control algorithm. However, the fairness problem in TCP is still not perfectly solved as we discussed earlier. Since TCP is completely ignorant of actual bandwidth needs of individual users, it attempts to equalize the bandwidth utilizations of all concurrent TCP sessions. Even if a desired level of goodput is explicitly known and given by some application, TCP lacks the capability of controlling the bandwidth allocation to achieve a stable and smooth goodput at the desired level. Consequently, problems may occur when the bandwidth is shared by network applications that have specific individual goodput requirements. More specifically, TCP typically provides more than needed throughput under low traffic, and "underflows" when the network is heavily loaded, thereby missing the target goodput most of the time.

The dynamics of AIMD congestion control algorithm used by TCP is another reason TCP is not well suited for providing stable goodput. In addition to the phase "slow start", in which the congestion window exponentially increases until it reaches *ssthresh*, TCP's AIMD congestion control process consists of two main phases: in the congestion avoidance phase, the congestion window size increases linearly until congestion is indicated by either three duplicate acknowledgements or retransmission timeout, which often results in throughput much higher than needed, especially in over-provisioned networks; while in the congestion detection and lost packet retransmission phase in the presence of burst losses, TCP drastically backs off the transmission rate by reducing congestion window size to typically half of the previous size, which very likely pulls the throughput significantly below the desired level. In practice, many network applications prefer a slow-but-uniform delivery rate to a fast-but-jumpy one.

In this chapter, we discuss the application of a dynamic version of the classical Robbins-Monro method [Wasan69] to the transport control for a desired level of goodput at the

destination node, and show its convergence under very general conditions, which is justified later using Internet experimental measurements. Our implementation of transport control based on UDP provides very stable and robust throughput over the Internet under various traffic conditions.

4.2 Problem Formulation

We consider a problem of transport control for stabilizing a transport stream from a source node (sender) *S* at a target throughput level at a destination node (receiver) *D* over a wide area network, typically the Internet, in presence of dynamically changing background traffics.

A message made up of a number of data packets is sent from source node *S* to destination node *D*. Similar to TCP, for every successfully received data packet, the receiver *D* sends an acknowledgement back to the sender *S*. Both data packets and acknowledgements may be delayed or lost due to a variety of reasons, like buffer occupancy levels at intermediate routers and end hosts. The lost data packets are infered and retransmitted in some way to guarantee the transmission reliability. The goodput $g_S(t)$ at the sender *S* and goodput $g_D(t)$ at the receiver *D* are calculated as the total number of received acknowledgments and data packets divided by the time elapsed, respectively. The transport control loop for throughput stabilization is illustrated in Figure 4.1.

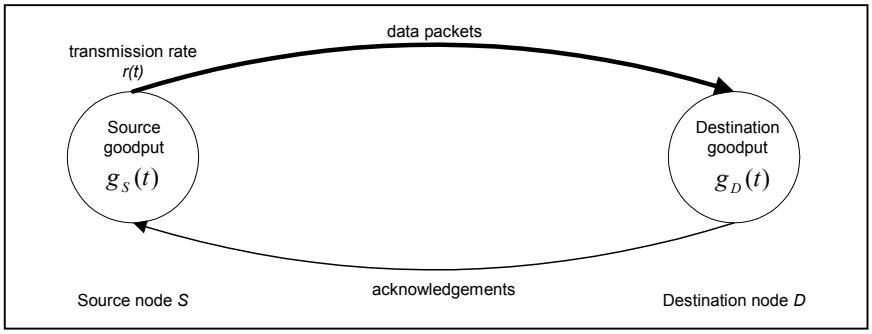


Figure 4.1 Transport control loop for throughput stabilization

The transport control objective is to achieve a constant target goodput g_D^* at the receiver *D* by dynamically controlling the transmission rate $r_S(t)$ (or simply $r(t)$) at the sender *S* through the adjustment made on either congestion window or sleep time.

Based on our network transport control model, the source transmission rate $r(W_c, T_s)$, upper limited by the capacity of Network Interface Card (NIC) and channel bandwidth, is determined by the combination of congestion window size W_c and sleep time T_s between the bursts of sending two adjacent full congestion windows. From the network traffic analysis in previous chapter, like the source goodput $g_S(\cdot)$, the destination goodput $g_D(\cdot)$ is also non-linearly correlated with the sending rate $r(t)$.

The main and critical difference, however, is that $g_S(\cdot)$ is a function of $r(t)$, but the relationship between $g_D(\cdot)$ and $r(t)$ is more complicated:

(1) Due to the randomness in network traffic, $g_D(\cdot)$ is a random variable at a fixed rate $r(t) = r$.

(2) The inherent non-linearity due to the presence of router buffers and policies, host buffers, and the interaction between the NIC and host CPU, results in a non-linear relationship between $r(t)$ and $g_D(\cdot)$.

In this section, we assume that the destination goodput response regression $M_D(r) = E[g_D(t) \mid r_S(t) = r] = \int g_D G(dg_D, r)$ (i.e. Equation (3.2)) is locally monotonic at the target goodput $g_D(t) = g_D{}^*$ such that there exists a target source rate r^* satisfying $M_D(r^*) = g_D{}^*$, and $M_D(r) > g_D{}^*$ if $r(t) > r^*$, and $M_D(r) < g_D{}^*$ if $r(t) < r^*$. The assumption validity will be further discussed later in this chapter.

Informally, by maintaining $r(t) = r^*$ we would achieve average goodput of $g_D{}^*$, and an increase (decrease) in r^* will result in an increase (decrease) in $M_D(r^*)$. Computing r^* however is difficult based on the information about the packet transmissions and acknowledgements alone.

4.3 Newton-Raphson Method

We first consider a simple case where the goodput response $g_D(r)$ is a deterministic function of sending rate *r*. Now the desired destination goodput problem is the root of a function, which in this case, may be rewritten as the original destination goodput function $g_D(r)$ minus a constant (i.e. the desired level $g_D{}^*$). If $g_D(r)$ were known, fixed and continuously differentiable as ideally characterized in [LPD02, Kelly99], then the problem would be a classical one in numerical analysis and the well-known Newton's method (also called Newton-Raphson method) could be used to produce accurate results in a few iterations using the recursive procedure:

$$r_{n+1} = r_n - \frac{1}{g_D{}'(r_n)} \cdot [g_D(r_n) - g_D{}^*], \tag{4.1}$$

where $g_D{}'(\cdot) = \frac{dg_D(r)}{dr}$. As illustrated in Figure 4.2, procedure (4.1) generates a sequence of root approximations, which converge to the actual root *r** linearly.

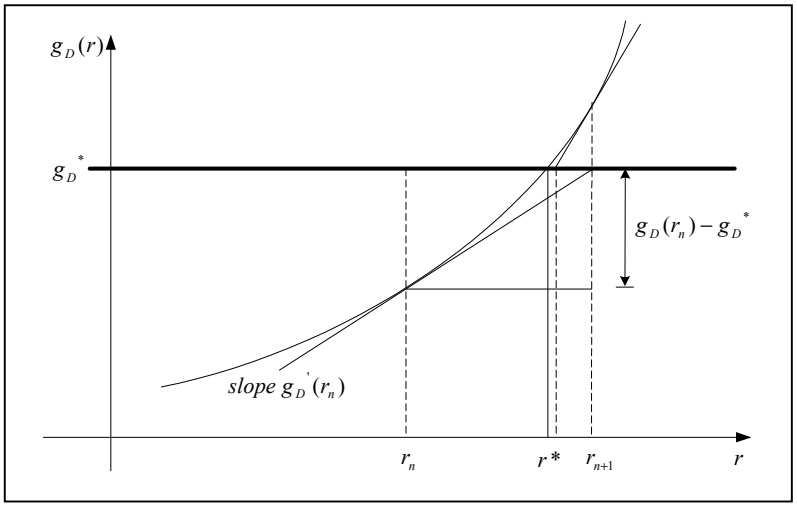


Figure 4.2 Newton method for a known and continuously differentiable function

We can show that Newton's method has the tendency to approximately double the number of digits of accuracy with each successive step [FB98]. Suppose that the second derivative of function $g_D(r)$ exists on an interval containing both r and the approximation r_n. The function $g_D(r)$ can be expanded at r_n and evaluated at the actual root r^* using Taylor polynomial:

$$g_D(r^*) = g_D(r_n) + g_D'(r_n)(r^* - r_n) + \frac{g_D''(r_n)}{2!}(r^* - r_n)^2 + \cdots = g_D^* \tag{4.2}$$

If the second derivative of function $g_D(r)$ is bounded by some finite constant, then we have:

$$g_D(r_n) + g_D'(r_n)(r^* - r_n) + \Theta(e_n^2) = g_D^* \tag{4.3}$$

where error e_n is the absolute distance between the actual root r^* and the n-th approximation r_n. Divide both sides of Equation (4.3) by the first derivative of function $g_D(r)$ evaluated at r_n and rearrange it:

$$r^* - r_n + \frac{g_D(r_n) - g_D^*}{g_D'(r_n)} = -\frac{O(e_n^2)}{g_D'(r_n)} \tag{4.4}$$

Plug $r_n = r_{n+1} + \frac{g_D(r_n) - g_D^*}{g_D'(r_n)}$ as given in the Newton iterative procedure into Equation (4.4):

$$r^* - r_{n+1} = -\frac{O(e_n^2)}{g_D'(r_n)} \tag{4.5}$$

Equation (4.5) shows that the error $e_{n+1} = | r^* - r_{n+1} |$ of the $(n+1)$-th approximation r_{n+1} is in the order of the square of the error e_n of the n-th approximation r_n.

4.4 Classical Robbins-Monro Stochastic Approximation

Unfortunately, the recursive procedure given by Newton's method cannot be directly applied to compute target rate r^* in network environments because the above assumptions about the goodput response are ideal and dramatic simplifications of the real network behavior and their validity is under question. The difficulties of the problem arise from the following concerns, which are primarily caused the stochastic nature of network traffic.

(1) The destination goodput $g_D(t)$ is not known at source and can only be estimated based on acknowledgments.

(2) Since $g_D(t)$ is random, the derivative $g_D'(\cdot)$ does not exist in general.

(3) There are various types of randomness involved in the data transmission over network as well as its performance measurements, due to which, $\hat{g}_D(t)$ will be continuously changing even if sending rate $r(t)$ is fixed. Consequently, $r(t)$ must be appropriately and continuously adjusted to maintain stable $g_D(\cdot)$.

In reality, the goodput response is non-smooth, non-differential, nonlinear, and its exact form is essentially unknown. The traffic measurement and analysis we conducted in previous chapter have clearly shown this point. In most cases, the only available

information in the end-to-end network transport control process is the "noise corrupted" goodput or loss rate estimates at the source node. This motivates us to explore new ways of transport control based on stochastic approximation methods. The classical Robbins-Monro Stochastic Approximation (RMSA) method enables us to compute the target rate using goodput estimation only.

Let g^* be the target goodput to be achieved at the destination. To compute the sending rate $r(t)$ needed to achieve g^*, the classical RMSA method uses the following updating scheme:

$$r_{n+1} = r_n - \varepsilon_n \left[\hat{g}(r_n) - g^* \right] \tag{4.6}$$

where $\hat{g}(r_n)$ is a "noisy" estimate or observation of the value of the unknown function $g(\cdot)$ at time step n, and ε_n is an appropriate positive real-valued sequence satisfying

$$\varepsilon_n > 0, \quad \varepsilon_n \to 0, \quad \sum_n \varepsilon_n = \infty \tag{4.7}$$

The procedure defined in Equation (4.6) is guaranteed to converge if the starting point r_0 is selected to be sufficiently close to the target rate r^* under fairly general conditions. The difference between a numerical approach and a stochastic approximation is obvious by comparing the RMSA recursive procedure defined by Equation (4.6) with Newton's method defined by Equation (4.1). The differentiability of function g is required in Newton's method, but not in RMSA method. An intuitive interpretation about RMSA method is that at each step the control adjustment direction is guided by the difference between the current observation $\hat{g}(r_n)$ and the explicitly given target level g^*, and the adjustment step size is regulated by the coefficient ε_n, which decreases as time step n goes to infinity but at a certain minimum rate.

Instead of using one single observation $\hat{g}(r_n)$ at time step n, one could run the testing with the same control parameters for a certain number of times and use the arithmetic mean of the observations to estimate the corresponding function value. Taking excessive measurements at each step obtains a better estimate and is preferable in offline computation. For some efficiency-critical applications such as network transport control, we would just take one single observation in order to meet its real-time control requirement.

As we see from Equation (4.7), the constraints on the coefficient sequence are actually quite loose. However, the choice of the sequence ε_n is considerably critical to the effectiveness or even validness of the algorithm. For instance, in our source rate control problem, a badly selected coefficient could produce a totally unreasonable sending rate (less than 0 or larger than the bandwidth), which may result in a schizophrenic behavior in the control system. How to choose an appropriate sequence for the application under study still remains a large issue. In most practical applications, the coefficients are empirically determined, so is the case in other stochastic approximation methods.

4.5 Dynamic RMSA for Desired Destination Goodput

The stochastic nature of the network traffic makes the achieved goodput vary with time even if the congestion window W_c and sleep time T_s are fixed. The relation between sending rate and goodput is of a very complicated form and essentially unknown. For a given sending rate, we are not able to obtain its accurate goodput but "noise corrupted" observations, which, together with the non-differentiability of the goodput response function, make the Newton's method invalid in the case of network traffic. On the other hand, we cannot simply apply the classical RMSA method either to the transport control problem because of two issues addressed below.

Firstly, the desired level of goodput g_D^* we aim to achieve is usually the goodput $g_D(t)$ at the destination node *D*. However, the end-to-end transport control is only performed at the sender side, which has no direct access to the destination goodput $g_D(t)$, but the source goodput $g_S(t)$. In other words, the observations $\hat{g}_S(t)$ made at the source node *S* are the estimates of $g_D(t)$. Recall that the source goodput is computed by dividing the total number of acknowledgements received so far by the time elapsed. The sum of loss rate of packets from *S* to *D* and acknowledgements from *D* to *S* is $r(t) - \hat{g}_S(t)$. If we assume that the loss process is symmetric in two-way directions, the data packets in the forward direction and the acknowledgements in the backward direction have the same loss rate $[r(t) - \hat{g}_S(t)]/2$. Then the source goodput observation can be estimated by:

$$\hat{g}_S(t) = g_D(t) - [r(t) - \hat{g}_S(t)]/2 \tag{4.8}$$

or equivalently

$$\hat{g}_S(t) = 2g_D(t) - r(t) \tag{4.9}$$

Recall that the regression function $M_D(r)$ defined in Equation (3.2) is the expected destination goodput conditioned on source sending rate *r*, $M_D(r) = E[g_D(t) \mid r_S(t) = r] = \int g_D G(dg_D, r)$. We have the expected source goodput $M_S(r)$ conditioned on both source sending rate *r* and expected destination goodput $M_D(r)$:

$$M_S(r) = E[\hat{g}_S(t) \mid r_S(t) = r,\ g_D(r) = M_D(r)] = 2M_D(r) - r \tag{4.10}$$

Obviously, the expected source goodput has a similar shape to the expected destination goodput under the circumstance that the low sending rate *r* causes very few packet losses, which implies $M_D(r) \approx r$. Since the only available information is on the source node, we assume that a source goodput $\hat{g}_S^*$ is observed with respect to the desired destination goodput g_D^*, which is intentionally chosen within the initial monotonically increasing part of $M_D(r)$ with low sending rates and very few packet losses. We can achieve the target source-sending rate r^* such that

$$M_S(r) < \hat{g}_S^* \ for\ r < r^* \quad \text{and} \quad M_S(r) > \hat{g}_S^* \ for\ r > r^* \tag{4.11}$$

Combining Equation (4.9) and (4.10), it follows that

$$M_D(r) < g_D^* \ for\ r < r^* \quad \text{and} \quad M_D(r) > g_D^* \ for\ r > r^* \tag{4.12}$$

Equation (4.12) serves as one of the assumptions for our later convergence proof. From now on, we use a general notation $g(r)$ to represent the goodput function of sending rate r.

Secondly, the classical RMSA method deals with random processes with static shapes or profiles. In other words, the goodput regression function *M(r)* does not vary with time. This requirement may not be always satisfied in the context of computer networks, especially during some periods of the year when special events occur. Also, the network traffic analysis conducted earlier clearly shows that the profile of the destination goodput is quite stable based on the measurements collected on normal days at the timescale of hours, days, weeks, and even months. For most data transfer applications over network, its duration of transmission process is usually within seconds, minutes, hours, and rarely days. Nevertheless, we shall investigate the unmodified dynamic RMSA in the presence of profile variation trend.

As Figure 4.3 shows, suppose there is a unique true solution of a dynamically changing target sending rate, denoted by θ_n to the desired goodput equation $g_n(r) = g*$ at time step n, where the unknown goodput function $g_n(r)$ has time-varying mean value in the form of regression function $M_n(r)$ as defined in Equation (3.2).

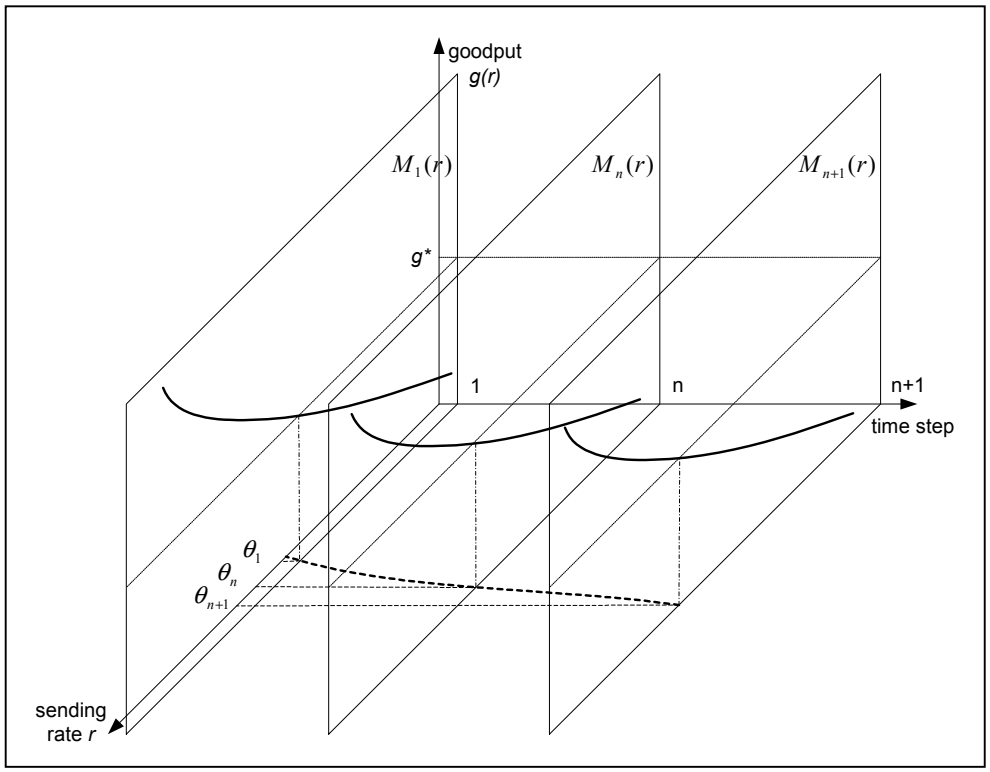


Figure 4.3 Time-varying goodput response regression in dynamic RMSA method

We further assume that the variation rate of target sending rate θ is constrained by:

$$\theta_{n+1} - \theta_n = O(n^{-\omega}) \tag{4.13}$$

We use the following recursive procedure to approximate the true position of the optimum solution:

$$r_{n+1} = r_n - \frac{a}{n^\alpha}(g_n - g^*) \qquad a > 0, \frac{1}{2} < \alpha < \min(1, \omega) \tag{4.14}$$

where g_n is a random variable such that:

$$E[g_n \mid r_i, g_i, r_n, i < n] = M_n(r_n) \qquad \text{for all } n \tag{4.15}$$

$$Var[g_n \mid r_i, g_i, r_n, i < n] \le \sigma^2 \qquad \text{for all } n \tag{4.16}$$

Recursive procedure defined by Equation (4.14) can be rewritten as:

$$r_{n+1} = r_n - \frac{a}{n^\alpha}(M_n - g^*) - \frac{a}{n^\alpha}(g_n - M_n) \qquad a > 0, \frac{1}{2} < \alpha < 1 \tag{4.17}$$

where the noise terms $(g_n - M_n)$, denoted by δM_n, are known as *martingale differences*. Equation (4.15) describes an important property of the stochastic process — martingale difference property, which together with the fact that the step sizes $\varepsilon_n = \frac{a}{n^\alpha}$ decrease to zero as *n* goes to infinity, is the key point to guarantee convergence with probability one.

4.6 Source Control Through Congestion Window Adjustment

As we discussed above, the source-sending rate can be controlled through the adjustment of either congestion window or sleep time. In this section we fix sleep time and dynamically adjust congestion window to stabilize the goodput at a desired level in the presence of a time-varying goodput response regression.

Recall that Equation (3.1) describes how the congestion window and sleep time together determine the source-sending rate: $r_S(t) = \dfrac{1.0}{\dfrac{T_s(t)}{W_c(t)} + \dfrac{1.0}{BW}}$, from which we have:

$$W_c(t) = \frac{T_s(t)}{\dfrac{1.0}{r_S(t)} - \dfrac{1.0}{BW}} \tag{4.18}$$

Consider that the sleep time is fixed at $T_s(t) = T_s$, and the desired goodput level is chosen within the initial monotonically increasing part of goodput response regression and hence the sending rate is a very small portion of the system bandwidth. We may rewrite Equation (4.18) in an approximate form:

$$W_c(t) \approx \frac{T_s}{\dfrac{1.0}{r_S(t)}} = T_s \cdot r_S(t) \tag{4.19}$$

It follows that the sending rate can be also approximated as:

$$r_S(t) \approx \frac{W_c(t)}{T_s} \tag{4.20}$$

Once the sending rate is updated at time step *n*+1 according to Equation (4.14), we apply the new congestion window $W_{c,\,n+1}$ as calculated below using Equation (4.19) and Equation (4.20):

$$W_{c,n+1} = T_s \cdot r_{S,n+1} = T_s \cdot [r_{S,n} - \frac{a}{n^{\alpha}} \cdot (g_n - g^*)]$$
$$= T_s \cdot [\frac{W_{c,n}}{T_s} - \frac{a}{n^{\alpha}} \cdot (g_n - g^*)] \quad (4.21)$$
$$= W_{c,n} - \frac{a \cdot T_s}{n^{\alpha}} (g_n - g^*)$$

Note that the only difference between the recursive procedure of sending rate in Equation (4.14) and the recursive procedure of congestion window in Equation (4.21) is the numerator of the step size adjustment coefficient. Since the sleep time is a real-valued constant T_s, if we choose an appropriate coefficient $a' = a \cdot T_s$ that incorporates T_s in Equation (4.21), the recursive procedure performed on congestion window will have the exactly same effect on the sending rate. Therefore, the convergence proof for the control system using the recursive procedure of sending rate can be also applied to the system based on the congestion window adjustment.

4.7 Source Control Through Sleep Time Adjustment

Besides congestion window, the source rate is also controllable through sleep time, i.e. the inter-cwin delay. In this section we fix congestion window and dynamically adjust sleep time to stabilize the goodput at a desired level in the presence of a time-varying goodput response regression.

From Equation (3.1), we obtain the expression for sleep time $T_s(t)$ in terms of congestion window $W_c(t)$ and sending rate $r_S(t)$ as follows:

$$T_s(t) = W_c(t)(\frac{1.0}{r_S(t)} - \frac{1.0}{BW}) \quad (4.22)$$

Again, since the desired goodput is targeted at a level very much below the system bandwidth, we may rewrite Equation (4.22) in the following approximate form with a fixed congestion window W_c:

$$T_s(t) \approx \frac{W_c}{r_S(t)} \quad (4.23)$$

It follows that the source rate is approximately determined by the ratio of fixed congestion window and varying sleep time:

$$r_S(t) \approx \frac{W_c}{T_s(t)} \quad (4.24)$$

At time step n+1, the new source rate updated using recursive procedure defined in Equation (4.14) is practically realized by adjusting the sleep time as follows:

$$T_{s,n+1} = \frac{W_c}{r_{S,n+1}} = \frac{W_c}{r_{S,n} - \frac{a}{n^{\alpha}} \cdot (g_n - g^*)}$$
$$= \frac{W_c}{\frac{W_c}{T_{s,n}} - \frac{a}{n^{\alpha}} \cdot (g_n - g^*)} \tag{4.25}$$
$$= \frac{1.0}{\frac{1.0}{T_{s,n}} - \frac{a/W_c}{n^{\alpha}} \cdot (g_n - g^*)}$$

We have the following recursive procedure for sleep time by rearranging Equation (4.25):

$$\frac{1.0}{T_{s,n+1}} = \frac{1.0}{T_{s,n}} - \frac{a/W_c}{n^{\alpha}} \cdot (g_n - g^*) \tag{4.26}$$

If we define a new sleep time variable as $T_{s,n}{}' = \frac{1.0}{T_{s,n}}$ and a new step size adjustment coefficient $a' = \frac{a}{W_c}$, Equation (4.26) then can be rewritten as:

$$T_{s,n+1}{}' = T_{s,n}{}' - \frac{a'}{n^{\alpha}} \cdot (g_n - g^*) \tag{4.27}$$

Compare the recursive procedures defined in Equation (4.14) for source rate and Equation (4.27) for newly defined sleep time, which is the inverse of the original sleep time. It is not difficult to see that both recursive procedures are in the exactly same form and hence the adjustment made on new sleep time parameter has the same effect on the source-sending rate if an appropriate value is chosen for the new coefficient a' in Equation (4.27). By the same token, the convergence of the system using the recursive procedure of sending rate also guarantees the convergence of the system through sleep time adjustment.

4.8 Convergence of Dynamic RMSA

We are concerned with the convergence of the network transport control system using the proposed dynamic RMSA method because the achieved goodput must be stabilized at a certain desired level. The following convergence theorem is provided for the control system based on source rate adjustment. However, as we discussed above, the convergence proof is equivalently applicable to the case where the source rate adjustment is carried out through either congestion window or sleep time.

Theorem 4.1: The source rate r_n updated by the recursive procedure of Equation (4.14), i.e.

$$r_{n+1} = r_n - \frac{a}{n^{\alpha}}(g_n - g^*) \qquad a > 0, \frac{1}{2} < \alpha < \min(1, \omega)$$

for the desired goodput problem converges to a true solution θ_n in quadratic mean:

$$E[(r_n-\theta_n)^2]=\begin{cases} O(\frac{1}{n^{\alpha}}) & \text{if} \quad \omega \geq \frac{3}{2}\alpha, \\ O(\frac{1}{n^{2(\omega-\alpha)}}) & \text{if} \quad \omega < \frac{3}{2}\alpha, \end{cases} \tag{4.28}$$

under the following assumptions:

(1) The stochastic process in the network transport control system exhibits the martingale difference property as defined in Equation (4.15), i.e. $E[g_n \mid r_i, g_i, r_n, i<n] = M_n(r_n)$ for all n. As illustrated in Figure 4.4, the expectation of the goodput g_n with source rate r_n measured for a sufficient large number of times at time step n falls on the goodput response regression curve M_n. Note that the unknown goodput response regression is subject to dynamic change as time goes on.

(2) The conditional variance of goodput at each time step is constrained by Equation (4.16), i.e. $Var[g_n \mid r_i, g_i, r_n, i<n] \leq \sigma^2$. As illustrated in Figure 4.4, this condition implies that the goodput measurements corresponding to a fixed source rate vary in a certain limited range.

(3) The target source rate, or true solution varies with time due to the dynamically changing goodput response regression, but its variation rate is constrained by Equation (4.13), i.e. $\theta_{n+1}-\theta_n = O(n^{-\omega})$. Note that the exponent α of the step size adjustment coefficient used in the recursive procedure of Equation (4.14) is less than the exponent ω. This relationship implies that the adjustment made on the step size in the source rate updating formula is faster than the variation of true solutions.

(4) The initial goodput response regression satisfies Equation (4.12): $M_1(r)-g^* < 0$ for $r<\theta_1$ and $M_1(r)-g^* > 0$ for $r>\theta_1$. As illustrated in Figure 4.4, this condition implies that the initial goodput response regression is a monotonic increase function in the neighborhood of the point at the target source rate θ_1.

(5) The initial goodput response regression satisfies the following constraint:

$$K_0 \mid r-\theta_1 \mid \leq \mid M_1(r)-g^* \mid \leq K_1 \mid r-\theta_1 \mid \quad \text{for} \quad -\infty < r < +\infty \tag{4.29}$$

where the slope K_0 and K_1 are two real-valued constants. As illustrated in Figure 4.4, this condition implies that the shape of the initial goodput response regression in the first monotonic increasing phase is completely "sandwiched" in a region defined by two straight lines crossing at the target rate θ_1.

(6) The expectation of the square of the random variable r_1, i.e. the initial approximate solution is constrained by

$$E[r_1^2] < +\infty \tag{4.30}$$

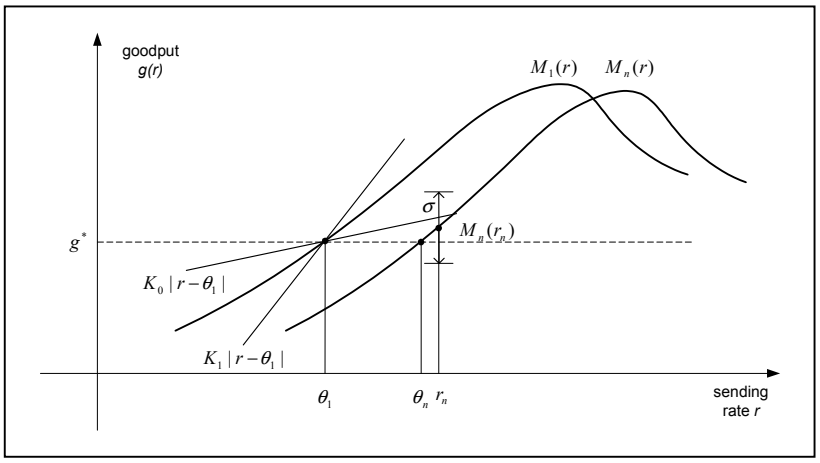


Figure 4.4 Illsutration of assumptions for convergence proof

Proof of Theorem 4.1:

The extensive performance measurements collected over a time span of more than six months show that the goodput response exhibits a very similar two-phase pattern and its overall profile remains stable for most of time in a year. Especially, the goodput response during the first phase monotonically increases with little variation. The observation data makes good sense that the convergence conditions (1) to (5) are reasonable assumptions under normal network traffic conditions. Besides, if we carefully choose a starting point for the source rate control system, assumption (6) can also be validated. However, we may need to reconsider these assumptions if TCOU competes with aggressive protocols not using end-to-end congestion control. The participation of aggressive protocols incurs rapid change in the goodput response so that some conditions such as condition (3) may not hold in this special case.

The formal convergence proof mostly follows the one given in [Dupac65] except that we have a non-zero goodput requirement g^*. We present it here anyway for the sake of completeness.

Suppose that real numbers $\theta_n, n = 1, 2, 3, \cdots$, are true solutions to the desired goodput problem: $M_n(r) - g^* = 0$. If we set $M_1(r) = M(r)$, then $\theta_n, n = 1, 2, 3, \cdots$, are the unique roots of the problem equation:

$$M_n(r) - g^* = M(x - \theta_n + \theta_1) - g^* = 0 \tag{4.31}$$

From the constraints of monotonic increase around the true solution θ_1 and bounded regression shape region given in Equation (4.12) and (4.29) respectively, the initial goodput response regression function can be expressed as follows:

$$M_1(r) - g^* = M(r) - g^* = \mu(r - \theta_1) \quad \text{for} \quad -\infty < r < +\infty \tag{4.32}$$

where the slope μ is a quantity whose value depends on the source rate *r* and satisfies

$$K_0 \le \mu \le K_1 \tag{4.33}$$

According to Equation (4.31), we may express the dynamically changing regression function as follows:

$$M_n(r) - g^* = M(r - \theta_n + \theta_1) - g^* = \mu(r - \theta_n + \theta_1 - \theta_1) = \mu(r - \theta_n) \tag{4.34}$$

The conditional expectation of goodput difference $g_n - g^*$ between observation and desired level can be calculated using Equation (4.15), (4.34):

$$E[g_n - g^* \mid r_i, g_i, r_n, i < n] = M_n(r_n) - g^* = \mu(r_n - \theta_n) \tag{4.35}$$

From the property of the variance of a random variable X: $Var[X] = E[X^2] - (E[X])^2$, it follows that $E[X^2] = Var[X] + (E[X])^2$. Using Equation (4.16), (4.33), and (4.35), we obtain an upper bound for the conditional expectation of $(g_n - g^*)^2$:

$$\begin{aligned} E[(g_n - g^*)^2 \mid r_i, g_i, r_n, i < n] &= Var[g_n - g^* \mid r_i, g_i, r_n, i < n] + (E[g_n - g^* \mid r_i, g_i, r_n, i < n])^2 \\ &\le \sigma^2 + [\mu(r_n - \theta_n)]^2 \\ &\le \sigma^2 + K_2(r_n - \theta_n)^2 \end{aligned} \tag{4.36}$$

where constant $K_2 \ge K_1^{\,2}$.

Subtract both sides of the true solution variation rate constraint given in Equation (4.13) from both sides of the recursive procedure defined in Equation (4.14) respectively and square:

$$\begin{aligned} (r_{n+1} - \theta_{n+1})^2 &= [r_n - \theta_n - \frac{a}{n^\alpha}(g_n - g^*) + O(n^{-\omega})]^2 \\ &= (r_n - \theta_n)^2 + \frac{a^2}{n^{2\alpha}}(g_n - g^*)^2 + O(n^{-2\omega}) \\ &\quad - \frac{2a}{n^\alpha}(r_n - \theta_n)(g_n - g^*) + 2(r_n - \theta_n)O(n^{-\omega}) + \frac{2a}{n^\alpha}(g_n - g^*)O(n^{-\omega}) \end{aligned} \tag{4.37}$$

Now we take conditional expectations on both sides of Equation (4.37) and use Equation (4.33), (4.35), and (4.36) to enlarge the right side of the equation:

$$\begin{aligned} E[(r_{n+1} - \theta_{n+1})^2 \mid r_i, g_i, r_n, i < n] &\le (r_n - \theta_n)^2 + \frac{a^2\sigma^2}{n^{2\alpha}} + \frac{a^2 K_2}{n^{2\alpha}}(r_n - \theta_n)^2 + \frac{K_3}{n^{2\omega}} \\ &\quad - \frac{2aK_0}{n^\alpha}(r_n - \theta_n)^2 + \frac{K_4}{n^\omega}\mid r_n - \theta_n \mid + \frac{aK_5}{n^{\alpha+\omega}}\mid r_n - \theta_n \mid \\ &= (1 + \frac{a^2 K_2}{n^{2\alpha}} - \frac{2aK_0}{n^\alpha})(r_n - \theta_n)^2 + (\frac{K_4}{n^\omega} + \frac{aK_5}{n^{\alpha+\omega}})\mid r_n - \theta_n \mid + (\frac{a^2\sigma^2}{n^{2\alpha}} + \frac{K_3}{n^{2\omega}}) \end{aligned} \tag{4.38}$$

where coefficients K_3, K_4, and K_5 are appropriate constants.

Further enlarge the right side of Equation (4.38) by ignoring the terms with higher exponents of the denominator and taking the relation $\alpha < \omega$ into consideration:

$$E[(r_{n+1} - \theta_{n+1})^2 \mid r_i, g_i, r_n, i < n] \le (1 - \frac{K_6}{n^\alpha})(r_n - \theta_n)^2 + \frac{K_7}{n^\omega}\mid r_n - \theta_n \mid + \frac{K_8}{n^{2\alpha}} \tag{4.39}$$

where coefficients K_6, K_7, and K_8 are appropriate constants such that $K_6 < 2aK_0$, $K_7 > K_4$, and $K_8 > a^2\sigma^2$.

Take unconditional expectations on both sides of Equation (4.39) and use the inequality $E[|X|] \le \varepsilon + \frac{1}{\varepsilon} E[X^2]$ to estimate $E[|r_n - \theta_n|]$, where $X = r_n - \theta_n$ is a random variable with finite variance and we set $\varepsilon = \frac{1}{\delta n^{\omega-\alpha}} > 0$ for small δ:

$$\begin{aligned} E[(r_{n+1} - \theta_{n+1})^2] &\le (1 - \frac{K_6}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_7}{\delta n^{2\omega-\alpha}} + \frac{K_7 \delta}{n^\alpha} E[(r_n - \theta_n)^2] + \frac{K_8}{n^{2\alpha}} \\ &= (1 - \frac{K_6 - K_7\delta}{n^\alpha}) E[(r_n - \theta_n)^2] + (\frac{K_7}{\delta n^{2\omega-\alpha}} + \frac{K_8}{n^{2\alpha}}) \end{aligned} \tag{4.40}$$

Therefore we may further simplify the right side of Equation (4.40) under different conditions depending on the values of variable ω:

$$E[(r_{n+1} - \theta_{n+1})^2] \le \begin{cases} (1 - \frac{K_9}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_{11}}{n^{2\alpha}}, & \text{if} \quad \omega \ge \frac{3}{2}\alpha \\ (1 - \frac{K_9}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_{10}}{n^{2\omega-\alpha}}, & \text{if} \quad \omega < \frac{3}{2}\alpha \end{cases} \tag{4.41}$$

To show the convergence in quadratic mean, we apply Chung's Lemma 4 [Chung54], which is presented as follows, to Equation (4.41):

Chung's Lemma 4. Suppose that $\{b_n\}, n \ge 1$, is a sequence of real numbers such that for $n \ge n_0$,

$$b_{n+1} \le (1 - \frac{c_n}{n^s}) b_n + \frac{c'}{n^t}, \tag{4.42}$$

where $0 < s < 1, s < t, c_n > c > 0, c' > 0$. Then

$$\lim_{n \to \infty} b_n \le \frac{c'}{c} \frac{1}{n^{t-s}} \tag{4.43}$$

Consequently, we obtain the following results:

$$E[(r_{n+1} - \theta_{n+1})^2] = \begin{cases} O(\frac{1}{n^\alpha}), & \text{if} \quad \omega \ge \frac{3}{2}\alpha \\ O(\frac{1}{n^{2\omega-2\alpha}}), & \text{if} \quad \omega < \frac{3}{2}\alpha \end{cases} \tag{4.44}$$

Proof ends.

4.9 Implementations of TCOU

TCOU has been implemented in C++ on Linux. In this section, we discuss the implementation details of Transport Control Over UDP (TCOU). The fundamental transport control framework attempts to resolve most of the transmission reliability related issues encountered by TCP with different techniques. The source rate control for both goodput stabilization and goodput maximization in Chapter 6 is built on this fundamental TCOU framework.

4.9.1 Framework of TCOU for Throughput Stabilization

As an end-to-end control protocol, TCOU consists of a sender and a receiver. The TCOU sender reads message and sends it as a set of UDP datagrams to the destination, while the

TCOU receiver receives and acknowledges datagrams in their arriving order, which are then either saved to the local hard disk or forwarded to applications on a sequential basis. Once all the datagrams are received, the receiver persistently sends the sender a special datagram called ALLRCVD signifying the end of transmission.

The TCOU sender consists of two child threads: one is for sending datagrams, and the other is for receiving acknowledgements. The TCOU receiver has a relatively simple structure because the transport control is performed at the sender side. Besides the datagram receiving thread, a separate child thread is created in the TCOU receiver to save or forward on-line datagrams sequentially. The control flow diagrams of TCOU sender and receiver are illustrated in Figure 4.5 and Figure 4.6, respectively.

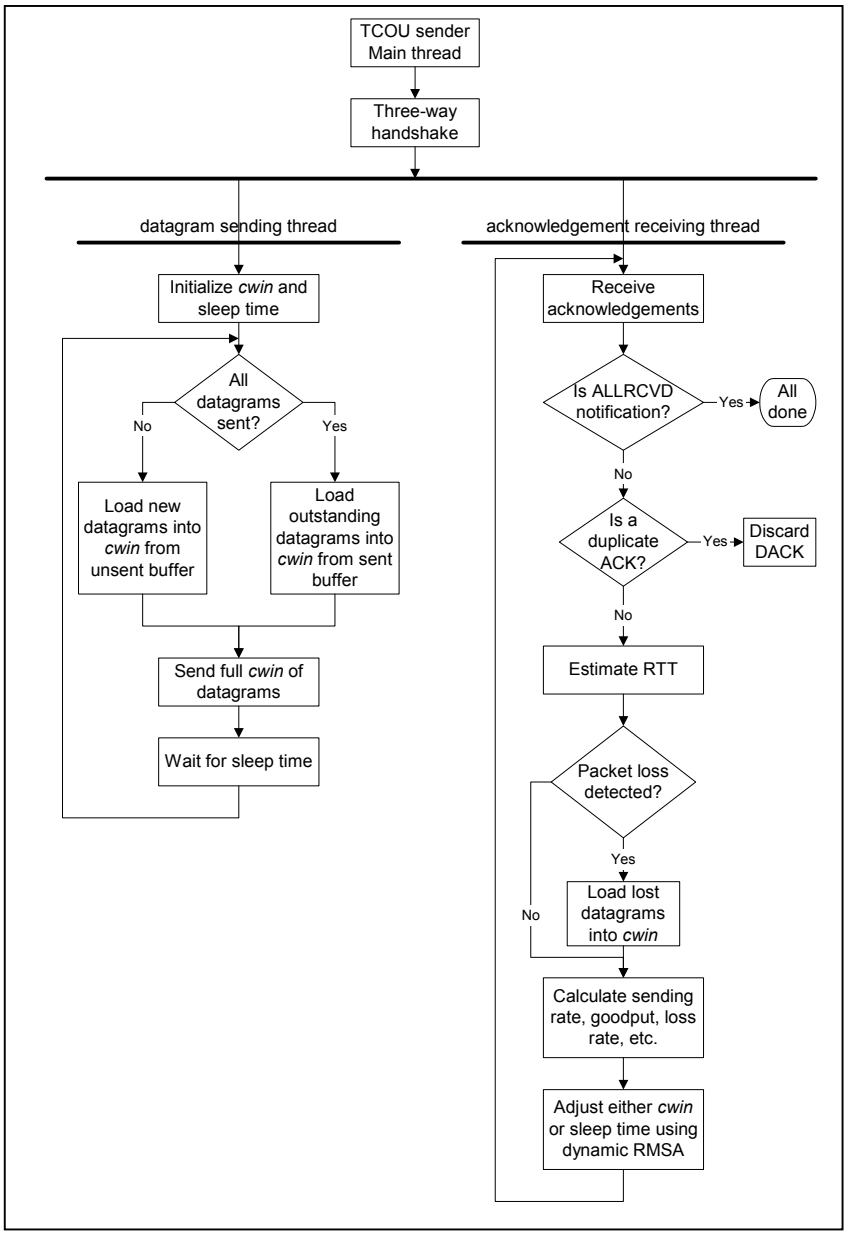


Figure 4.5 Control flow diagram of TCOU sender for goodput stabilization

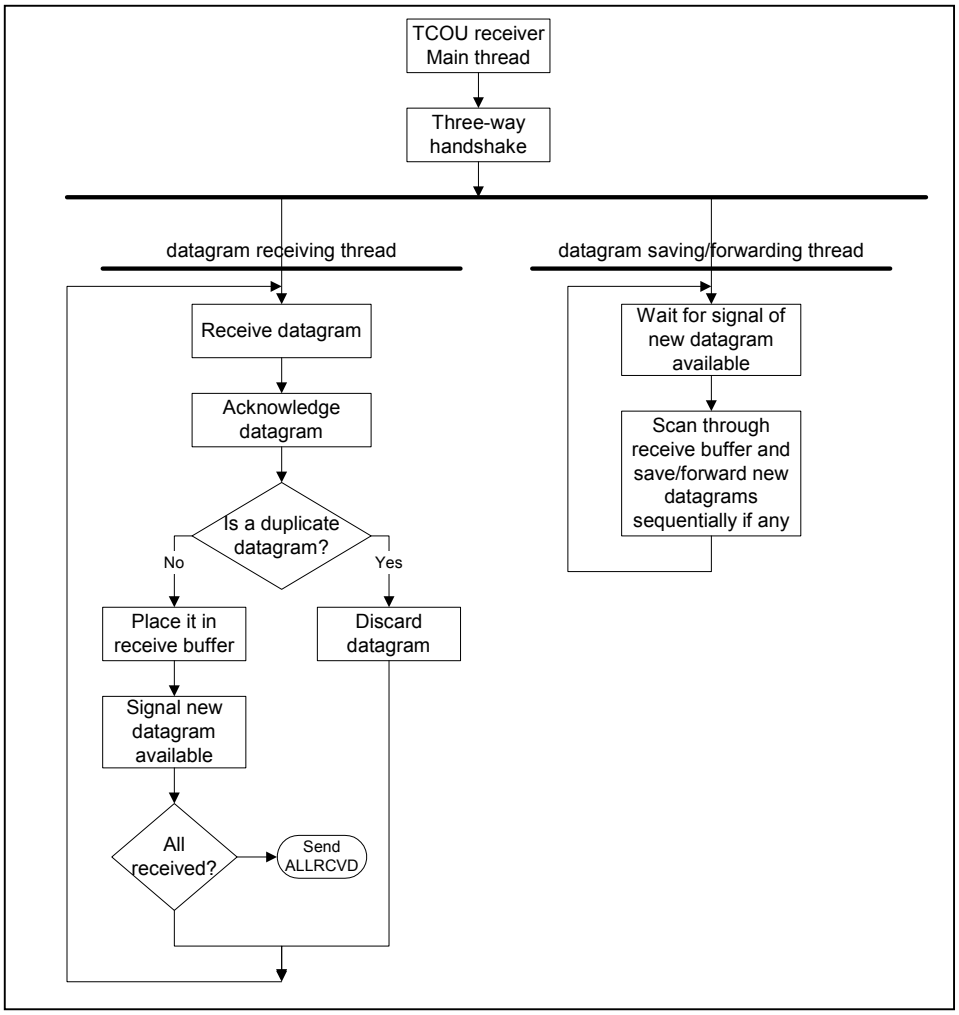


Figure 4.6 Control flow diagram of TCOU receiver

Rate control related activities on the right side in Figure 4.5, such as RTT estimation, packet loss detection, goodput and loss rate calculation, etc. are carried out in the acknowledgement receiving thread upon the arrival of each acknowledgement. The source rate is controlled by the dynamic RMSA method through on-line adjustment of either *cwin* or sleep time.

4.9.2 Datagram Sequencing and Acknowledging

In TCP, a continuous sequence number is assigned to every single byte so that a TCP segment is identified by the sequence number of its first byte of data and its size. In TCOU, the message is first divided into a set of UDP datagrams (usually MTU size minus system-defined IP and UDP header lengths). To reduce the user-defined header size, we use datagram sequencing instead of byte sequencing. Specifically speaking, every UDP datagram is assigned a continuous aggregate sequence number.

On the other hand, the acknowledgment of a segment in TCP is the very next expected sequence number, i.e. the sequence number of its last byte of data plus one.

Corresponding to the datagram sequencing, we acknowledge every single successfully received datagram in TCOU using its associated datagram sequence number.
The datagram sequencing and acknowledging technique makes it possible to implement a new flow control scheme based on a floating window instead of a conventional sliding window. The implementation details will be given later.

4.9.3 Three-way Handshake and Connection Termination

Like TCP, TCOU initiates a connection between a sender and a receiver using a three-way handshake. Firstly, the active client sends a SYN datagram to the passive server. Secondly, the server sends back to the client an ACK-SYN datagram upon receipt of the SYN datagram. Finally, the client acknowledges the receipt of ACK-SYN datagram by sending an ACK datagram to the server. Figure 4.7 shows the three-way handshake scheme implemented in TCOU, where ACK-ACK is sent by the server to the client to acknowledge the last ACK sent by the client. As a matter of fact, both ACK and ACK-ACK are not critical to the three-way handshake so that TCOU may complete the connection initiation if a preset timeout expires. The SYN datagram consists of three fields: message name, message size, and the number of datagrams.

```
while(ACK-SYN not received)
{
    send SYN datagram;
    read datagram in non-blocking mode;
    if(datagram is ACK-SYN)
    {
        while(ACK-ACK not received)
        {
            send ACK datagram;
            read datagram in non-blocking mode;
            if(datagram is ACK-ACK || Timeout)
                break;
        }
    }
}
```

(a) Active client

```
while(SYN not received)
{
    read datagram in blocking mode;
    if(datagram is SYN)
    {
        while(ACK not received)
        {
            send ACK-SYN datagram;
            read datagram in non-blocking mode;
            if(datagram is ACK || Timeout)
                break;
        }
}
```

(b) Passive server

Figure 4.7 Three-way handshake in TCOU: (a) Active client, (b) Passive server

On the contrary, the receiver plays an active role in the connection termination phase because it is informed of the total number of datagrams to be sent in the beginning of the connection and hence has the knowledge of whether all datagrams are received or not. As soon as the last datagram arrives (not necessarily the one with the last sequence number due to the implementation of a floating-window based flow control), the server persistently sends out ALLRCVD notification to the client until the ALLRCVD notification is acknowledged. Then both server and client terminate the connection and clean up all computer and communication resources it occupies.

4.9.4 Floating Window Based Flow Control

The flow control in TCP is provided through sliding window technique. The main characteristic of a sliding window is its continuity, which together with cumulative acknowledgement mechanism, facilitates the implementation of congestion control

strategies, but also imposes disadvantages on the network performance. As we discussed in detail in Chapter 2, with the limited information available from cumulative acknowledgements, a TCP sender can only learn a single lost packet per round trip time. Specifically speaking, the left edge of the sliding window (i.e. offered window: minimum of *cwnd* and *rwnd*) cannot advance until the oldest outstanding segment pointed by *snd_una* is acknowledged. That means that all other outstanding segments after the *snd_una* segment have to be held up in the sliding window waiting for the acknowledgement of the *snd_una* segment no matter whether or not they actually have successfully arrived at the destination. Therefore, there is some possibility that the whole sliding process is hindered by a single lost packet. More seriously, TCP may experience very undesirable performance if multiple packets are lost from one window of data.

The TCP Selective Acknowledgement (SACK) mechanism, combined with a selective repeat retransmission policy, is proposed to overcome these limitations [MMFR96]. By using SACK option, the receiving TCP is able to inform the sender of a list of data blocks that has been received. Each block, identified by its left and right edges, represents received bytes of data that are contiguous and isolated. The sender is ideally required to retransmit only the missing data segments to save transmission time and communication resources. However, due to the shortage of buffer space, the receiving TCP is permitted to discard data in its queue that has been reported in a SACK option, but not been acknowledged to the data sender yet. Consequently, the sender still needs to keep all outstanding segments in the sliding window even if they have already been SACKed. The policy of discarding at the receiver and holding at the sender the SACKed segments still guarantees the continuity of sliding window, but abates the benefit of using the TCP SACK option.

In TCOU, we implement a new flow control scheme based on a floating window, which is essentially a virtual window storing pointers to real datagrams. The full window of datagrams are transmitted every interval of sleep time. As indicated by its name, a discrete floating window is dividable into small chunks each of which “drifts” on the different parts of the sender buffer. The data structure of such a floating window is illustrated in Figure 4.8.

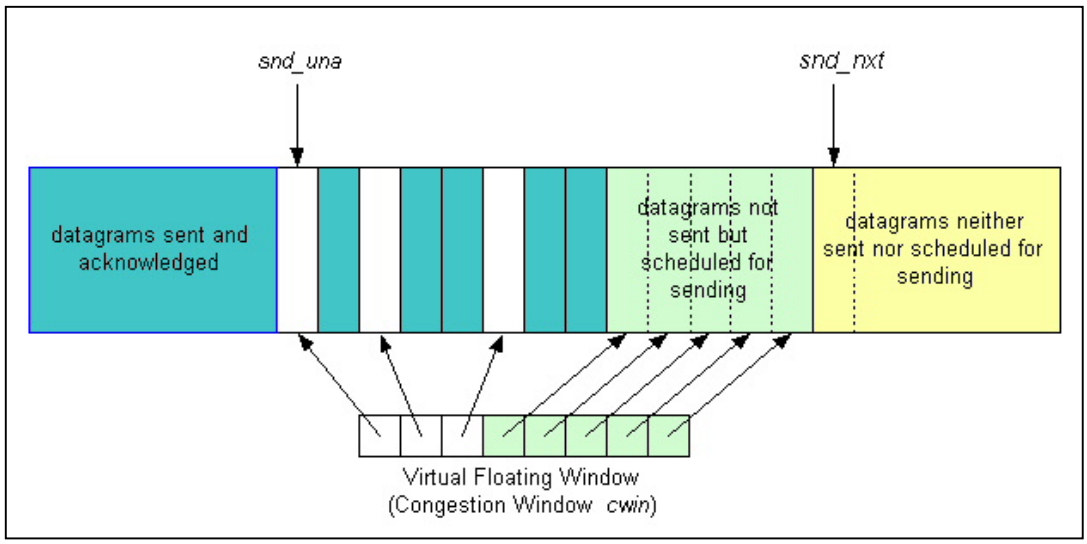


Figure 4.8 Floating window

Same as TCP sliding window depicted in Figure 2.4, the state variable *snd_una* points to the oldest outstanding datagram and *snd_nxt* points to the first datagram, which has neither been sent nor scheduled for sending. Figure 4.7 shows that the virtual floating window may consist of pointers to both retransmitted datagrams and newly scheduled datagrams. Apparently the floating window does not maintain window continuity and hence is able to effectively handle multiple lost packets from one window. Since the arrival of the acknowledgement of an out-of-order datagram means that this datagram has left the network and released the occupied resources, it is immediately removed from the sender buffer without waiting for holes before it to be filled and the window continues to move forward. The floating window allows us to accept any out-of-order datagrams and we only need to retransmit those datagrams that are really lost. Therefore, the floating window provides a more efficient flow control mechanism than the sliding window used by the traditional TCP.

4.9.5 Packet Loss Detection and Lost Packet Retransmission

A TCP acknowledgement carries the sequence number of the very next contiguous byte to be received. If segments arrive at the destination out of order, their acknowledgements having the same acknowledging sequence number are refereed to as duplicate acknowledgements (DACK). In TCP, the receipt of three DACK usually indicates the regular packet loss while in the case of severe network congestion the retransmission timeout is used.

In TCOU with flow control based on floating window, since every single datagram is individually sequenced and acknowledged, we shall apply different schemes of packet loss detection and lost packet retransmission. As in TCP Vegas, to effectively measure the network condition, the sender stamps the sending time of each datagram and estimates the round trip time (RTT) for each acknowledgement sent back by the receiver. The congestion detection algorithm described in Figure 4.9 is carried out upon the receipt of an acknowledgement.

```
read acknowledgement;
extract acknowledged datagram sequence number ackedDgSeqNo;
remove the datagram of ackedDgSeqNo out of the buffer;
if(ackedDgSeqNo == snd_una)
    move snd_una forward until the first outstanding datagram is found;
else if(ackedDgSeqNo > snd_una)
{
    for each outstanding datagram odg from snd_una to (ackedDgSeqNo – 1)
        if(odg is not in cwin && outstanding time of odg > 3 * RTT)        //packet loss
detected
            load odg into cwin;
}
```

Figure 4.9 Algorithm for packet loss detection

Note that the congestion window *cwin* in Figure 4.9 is the same as the virtual floating window and the "outstanding time" is the duration of the datagram being outstanding, or equivalently the time elapsed since the datagram is sent. The algorithm requires that all holes (outstanding datagrams) before the out-of-order acknowledgement are checked and

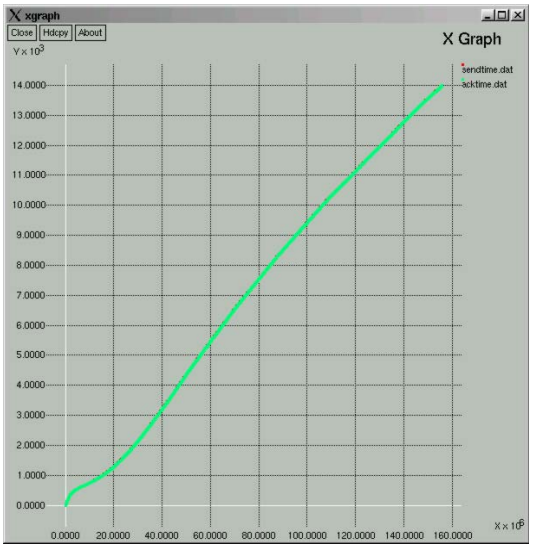
reloaded into the congestion window if they are not already in and have remained outstanding for at least three times RTT. Obviously, both retransmitted datagrams and newly scheduled datagrams from unsent buffer could be mixed in the same congestion window. The occupation percentage depends on the source rate and the network condition. If all acknowledgements arrive in order, we assume that the network is not congested so that no outstanding datagrams need to be retransmitted and a full window of datagrams is loaded only from the unsent buffer. In the other extreme case where the network is heavily loaded, the congestion window may be fully filled with the retransmitted datagrams. In any case, the full window of datagrams is sent for every interval of sleep time to maintain the designated source-sending rate.

4.10 Experimental Results of Throughput Stabilization

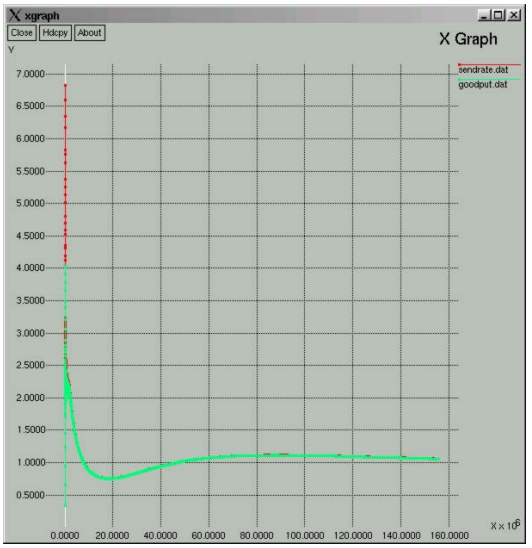
The throughput stabilization experiments are conducted between the host ozy4.csm.ornl.gov at ORNL and the host resource.rrl.lsu.edu at LSU. The adjustment is made on either congestion window or sleep time to meet various goodput requirements. We run on-host and off-host background network traffic such as HTTP, FTP, and SSH, etc., during the experiments to test the robustness of our methods.

Case 1. Target goodput = 1.0 Mbps, rate control through congestion window

In this case we attempt to achieve the desired goodput level of 1.0 Mbps. Two coefficients of the recursive procedure of the dynamic RMSA method are set as a = 0.8 and $\alpha = 0.8$. The starting point is arbitrarily selected at the point where *cwin* = 30 ms and sleep time = 100 ms. The source rate is controlled through the adjustment made on the congestion window only. A test message of 20 Mbytes is created on host ozy4.csm.ornl.gov and transferred to host resource.rrl.lsu.edu. Plot (a) in Figure 4.10 shows the curve of the datagram sending and acknowledging time vs. datagram sequence numbers, and Plot (b) shows the curve of the corresponding datagram acknowledging time vs. source sending rate and goodput.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers

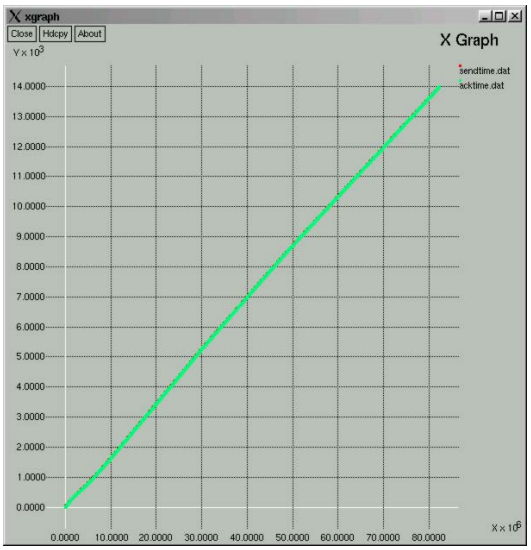


(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

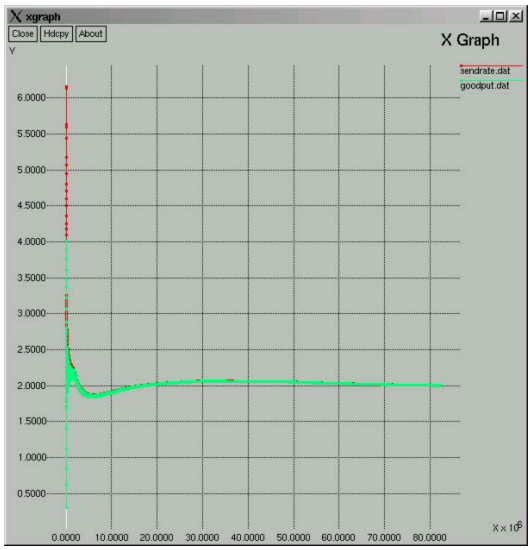
Figure 4.10 Desired goodput level = 1.0 Mbps, a = 0.8, $\alpha = 0.8$, adjustment made on congestion window

Case 2. Target goodput = 2.0 Mbps, rate control through congestion window

In this case the same network control settings in Case 1 are applied except that the new target goodput level is increased to 2.0 Mbps. The datagram sending and acknowledging time vs. datagram sequence numbers is plotted in Plot (a) of Figure 4.11 and the datagram acknowledging time vs. source rate and goodput is plotted in Plot (b).



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers



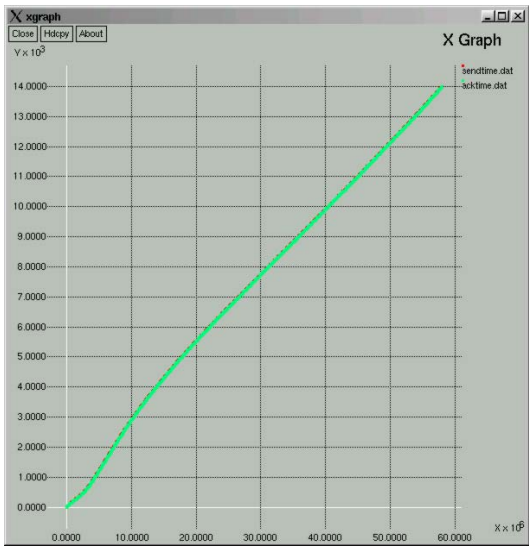
(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

Figure 4.11 Desired goodput level = 2.0 Mbps, $a = 0.8$, $\alpha = 0.8$, adjustment made on congestion window

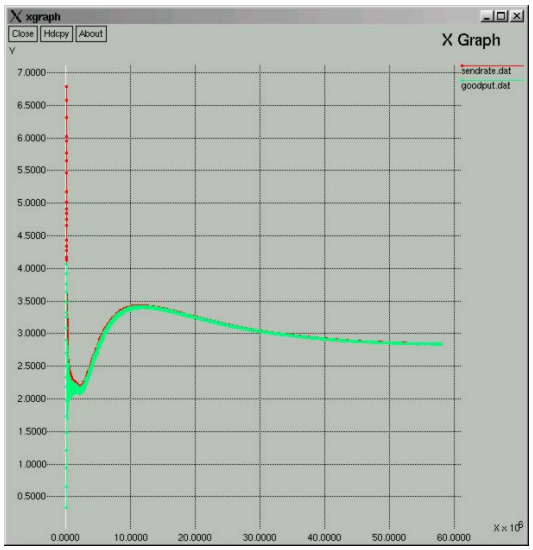
Compare Plot (b) in Figure 4.10 and Figure 4.11. It has been observed that the rate control process is stabilized faster in Case 2 than in Case 1. This is mostly due to the fact that the arbitrarily selected starting point is closer to the target level of 2.0 Mbps in Case 2. The rate control process may experience more oscillations when the target level is further increased as is in Case 3 below. Apparently, selecting an appropriate starting point around the given target level helps stabilize the control process and achieve the desired goodput quickly. Since the target source rate matches the desired goodput level with very little packet loss, Equation (3.1) can be used to find the pair of congestion window and sleep time for such a starting point.

Case 3. Target goodput = 3.0 Mbps, rate control through congestion window

Again in this case we use the same network control settings and further increase the desired goodput level to 3.0 Mbps. Two corresponding curves of the datagram sending and acknowledging time vs. datagram sequence numbers and the datagram acknowledging time vs. source sending rate and goodput are shown in Plot (a) and (b) in Figure 4.12, respectively.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers

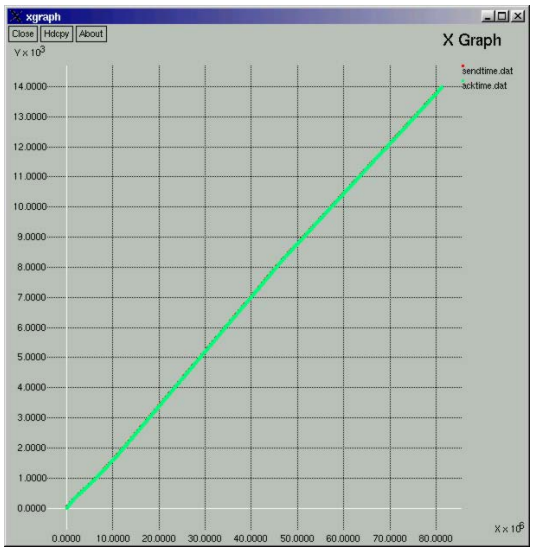


(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

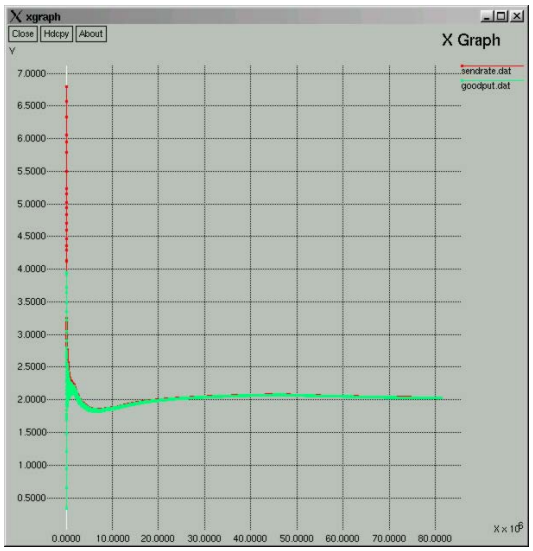
Figure 4.12 Desired goodput level = 3.0 Mbps, $a = 0.8$, $\alpha = 0.8$, adjustment made on congestion window

Case 4. Target goodput = 2.0 Mbps, rate control through sleep time

The rate control in the above three cases is conducted through the adjustment made on congestion window. We are also interested in examining the effect of sleep time adjustment on the network performance. In this case the source rate is controlled through sleep time and the target goodput is aimed at 2.0 Mbps. All other settings are the same as before.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers



(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

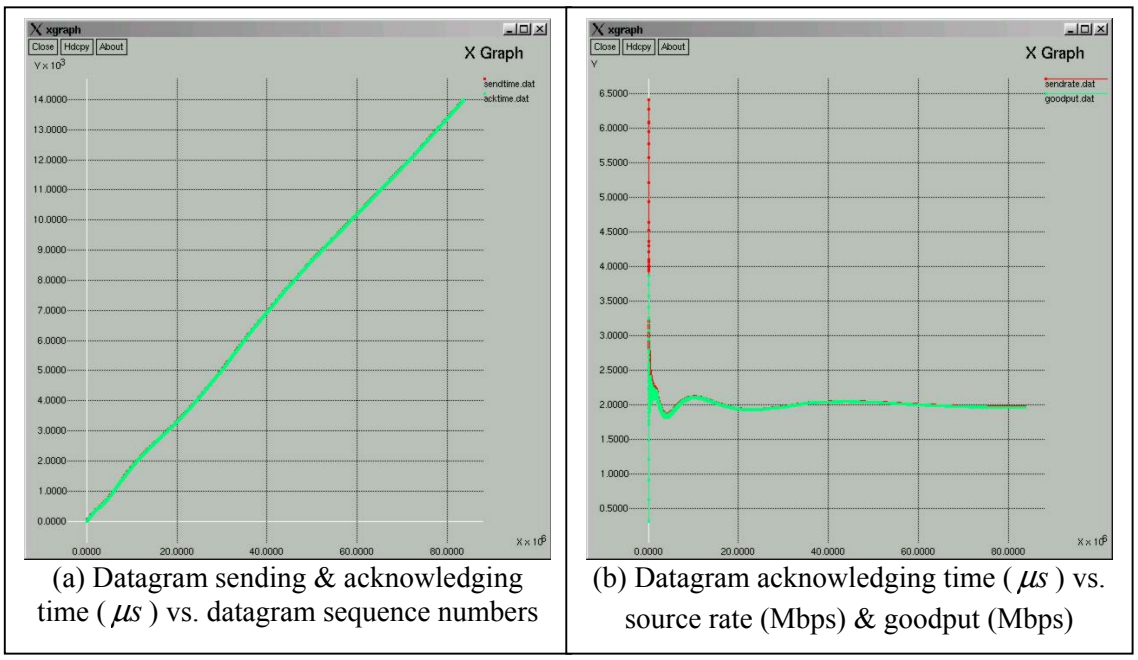
Figure 4.13 Desired goodput level = 2.0 Mbps, $a = 0.8$, $\alpha = 0.8$, adjustment made on sleep time

The datagram sending and acknowledging time vs. datagram sequence numbers is plotted in Plot (a) and the datagram acknowledging time vs. source rate and goodput is plotted in Plot (b) in Figure 4.13, respectively.

Note that Case 2 and Case 4 have all the same network settings but the rate control is carried out through adjustment on different objects. Based on the comparison between Figure 4.11 and Figure 4.13, we know that congestion window and sleep time have very similar behaviors of affecting the network performance. This observation justifies our statistical analysis results that both congestion window and sleep time have significant effects on the network transport control.

Case 5. Target goodput = 2.0 Mbps, rate control through sleep time, different coefficients

In this case we study the effects of different coefficients on the control process stabilization. The desired goodput level is still aimed at 2.0 Mbps, and all other network settings are the same as before. We set new coefficients a = 0.9 and $\alpha = 0.6$ in the recursive procedure of the dynamic RMSA method. Figure 4.14 shows the two curves of the datagram sending and receiving time vs. datagram sequence numbers and the corresponding datagram acknowledging time vs. source sending rate and goodput.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers

(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

Figure 4.14 Desired goodput level = 2.0 Mbps, a = 0.9, $\alpha = 0.6$, adjustment made on sleep time

Since coefficient a has a positive effect while coefficient α has a negative effect on the adjustment step size in the recursive procedure, the increase of a and decrease of α produce a bigger step size so that the stabilization process in this case must be subject to more intensive oscillations than Case 4. The plots in Figure 4.14 show exactly what we have anticipated.

According to the experimental results presented above, we have the following conclusions:

(1) The control process converges very reliably and quickly to the desired goodput level, which is reasonably chosen within the first phase, i.e. the initial monotonically increasing part of goodput response regression.
(2) Selection of the starting point affects the speed of convergence and intensity of oscillation. A suitable starting source rate is located around the target level and its corresponding congestion window and sleep time can be approximately calculated from Equation (3.1).
(3) Coefficients of the recursive procedure in TCOU for goodput stabilization also have some impact on the control process. In general, the larger step adjustment size the coefficients generate, the more intensively the control process oscillates. On the contrary, the smaller step adjustment size brings about smoother process but may cause slower convergence.
(4) The proposed method is robust against the presence of various normal TCP-based background traffic such as HTTP, FTP, SSH, etc.

CHAPTER 5 TCOU FOR GOODPUT MAXIMIZATION

5.1 Introduction

Computer network has undergone a tremendous evolution during the last twenty years. Especially, as driven by Moore's Law, the communication speed has significantly improved from the order of Kbps in the early stage of the network development to the order of Gbps widely available in the present backbone, and the increase tendency still persists. Nowadays, network researchers are not concerned about the bandwidth limit as much as before but more interested in achieving the maximum bandwidth utilization.

The current implementation of TCP uses AIMD congestion control algorithm, which is not well suited for high-bandwidth and long-delay links. According to Equation (1.9), the maximum sending rate of a TCP flow is only constrained by MSS, RTT, and packet drop rate. In other words, for a conformant TCP flow, high bandwidth does not necessarily guarantee high sending rate. Due to its conservative nature, TCP may result in poor bandwidth utilization as low as less than 10% in many cases. On the other hand, an extremely aggressive protocol may be able to seize as much as 90% of the path bandwidth. However, exclusively occupying the link by killing all the other participating sessions to achieve high bandwidth utilization is not a good solution either in the sense of fairness.

In this chapter, our research objective is to achieve high bandwidth utilization by fairly maximizing the individual throughputs from the overall perspective. The new transport control protocol is intended to work concurrently with other existing protocols and procure its own share of the available bandwidth as much as possible. In view of the unimodal property of the goodput response curve shown in Figure 4.4, we know that the maximum individual throughput can be achieved at the optimal sending rate with a low packet loss rate. However, there are three issues to be considered.

First of all, as we have pointed out in many places in this dissertation and will emphasize again here, none of the relationships between the control parameters and response variables in the network transport control system can be defined in a deterministic form. Particularly, the goodput measurements vary from run to run at the same source rate due to dynamically and continuously changing network conditions so that the goodput function in response to the source rate is not smoothly differentiable. In recent years, a great deal of research work has been focused on the maximization problem in solving a system of differential equations [Kelly99, LPD02]. All these theoretical works are based on some dramatic simplifications of the network behaviors by assuming the smoothness and differentiability of the utility or cost function. However, we shall now still explore suitable stochastic approximation methods to design practically useful transport control protocols since the observed stochastic nature of network traffic rules out any ideal derivative-based maximization (or minimization) methods.

Secondly, unlike the desired goodput problem, no explicitly given target value can be used here to guide the search process for optimal source rate. This concern motivates us to apply the stochastic optimization algorithms where the search direction is determined by the gradient approximations based on function measurements. Such algorithms do not require detailed modeling information about the relationship between control parameters and the objective function.

Finally, since the goodput response regression is subject to change in the presence of various background traffics, we also need to consider a time-varying version of the stochastic approximation method, which is chosen to solve the goodput maximization problem.

5.2 Classical Kiefer-Wolfowitz Stochastic Approximation

We wish to solve for the maximum of the "noise corrupted" goodput function over two dimensional parameter sets. It is assumed that the goodput regression function $E[g(T_s(t),W_c(t))] = M(T_s,W_c)$ in response to two control parameters, congestion window and sleep time, is continuously differentiable, but the exact forms of $g(\cdot)$ and $M(\cdot)$ are of stochastic nature and essentially unknown. The solution based on the classical Kiefer-Wolfowitz Stochastic Approximation (KWSA) method is a multi-variable recursive optimization procedure defined as:

$$\begin{bmatrix} W_{c,n+1} \\ T_{s,n+1} \end{bmatrix} = \begin{bmatrix} W_{c,n} \\ T_{s,n} \end{bmatrix} + a_n \begin{bmatrix} \hat{G}_{W_{c,n}}(W_{c,n}, T_{s,n}) \\ \hat{G}_{T_{s,n}}(W_{c,n}, T_{s,n}) \end{bmatrix} \tag{5.1}$$

where $a_n > 0$ represents a scalar gain coefficient. The gradient G of the goodput regression function is approximated by a "two-sided" finite difference:

$$\begin{aligned} \hat{G}_{W_{c,n}}(W_{c,n}, T_{s,n}) &= \frac{\hat{g}(W_{c,n} + c_n, T_{s,n}) - \hat{g}(W_{c,n} - c_n, T_{s,n})}{2c_n} \\ \hat{G}_{T_{s,n}}(W_{c,n}, T_{s,n}) &= \frac{\hat{g}(W_{c,n}, T_{s,n} + c_n) - \hat{g}(W_{c,n}, T_{s,n} - c_n)}{2c_n} \end{aligned} \tag{5.2}$$

where c_n is a small positive number. To guarantee convergence, the parameters a_n and c_n must satisfy the following criteria:

$$\lim_{n\to\infty} a_n = 0, \quad \lim_{n\to\infty} c_n = 0, \quad \sum_{n=1}^{\infty} a_n = \infty, \quad \sum_{n=1}^{\infty} \left(\frac{a_n}{c_n}\right)^2 < \infty \tag{5.3}$$

According to Equation (5.2), we illustrate the calculation of gradient approximation in Figure 5.1, where each component of the control parameters is perturbed one-at-a-time and the corresponding component of the gradient is formed by differencing the two measurements of the goodput function and then dividing by the difference interval. This "two-sided" gradient approximation approach is directly motivated from the definition of a gradient as a vector of partial derivatives [Spall94].

As Figure 5.1 shows, four measurements (two for each dimension) must be made in the classical KWSA method to move one step further from the current position. Consider that the network transport requires real-time control. Collecting four measurements online at each step may take too much time to satisfy the real-time performance requirement. Therefore, we shall explore other ways to reduce the number of measurements to be made for each step.

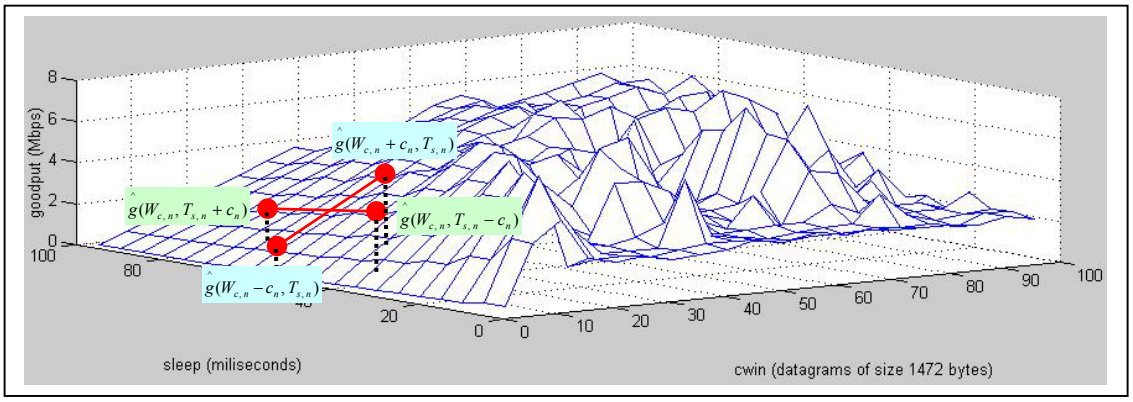


Figure 5.1 Two-sided gradient approximation in KWSA method

5.3 Dynamic Simultaneous Perturbation Stochastic Approximation

The well-known KWSA discussed in the preceding section needs $2p$ function measurements to approximate the gradient in a p-dimensional optimization space. The essential feature of Simultaneous Perturbation Stochastic Approximation (SPSA) is that it requires only two objective function measurements per iteration regardless of the dimension of the optimization problem. In contrast with KWSA, SPSA randomly perturbs all components of the control parameter set together in two separate directions to obtain two measurements, which are used to compute all components of the gradient. Although SPSA uses p times fewer function evaluations than KWSA, SPSA achieves the same level of statistical accuracy as KWSA does for a given number of iterations under reasonably general conditions [Spall92].

The gradient approximation of the goodput function based on SPSA is given by:

$$\hat{G}_{W_{c,n}}(W_{c,n}, T_{s,n}) = \frac{\hat{g}(W_{c,n}+\Delta_{W,n}c_n, T_{s,n}+\Delta_{T,n}c_n) - \hat{g}(W_{c,n}-\Delta_{W,n}c_n, T_{s,n}-\Delta_{T,n}c_n)}{2\Delta_{W,n}c_n}$$
$$\hat{G}_{T_{s,n}}(W_{c,n}, T_{s,n}) = \frac{\hat{g}(W_{c,n}+\Delta_{W,n}c_n, T_{s,n}+\Delta_{T,n}c_n) - \hat{g}(W_{c,n}-\Delta_{W,n}c_n, T_{s,n}-\Delta_{T,n}c_n)}{2\Delta_{T,n}c_n} \quad (5.4)$$

where the coefficient sequence $\{\Delta_{i,n}\}$ are independent and symmetrically distributed about 0 with finite inverse moments $E(|\Delta_{i,n}|^{-1})$ for all parameter components i and time steps n [Spall00]. A simple and theoretically valid choice for each component of sequence $\{\Delta_{i,n}\}$ is to use a symmetric Bernoulli ± 1 distribution with probability of $\frac{1}{2}$ for each outcome of either +1 or –1 [Spall98]. In the case of goodput maximization problem, we consider two components of the control parameter set, i.e. congestion window W and sleep time T, both of which use the same numerator to calculate their respective gradient components in Equation (5.4). The calculating process is illustrated in Figure 5.2.

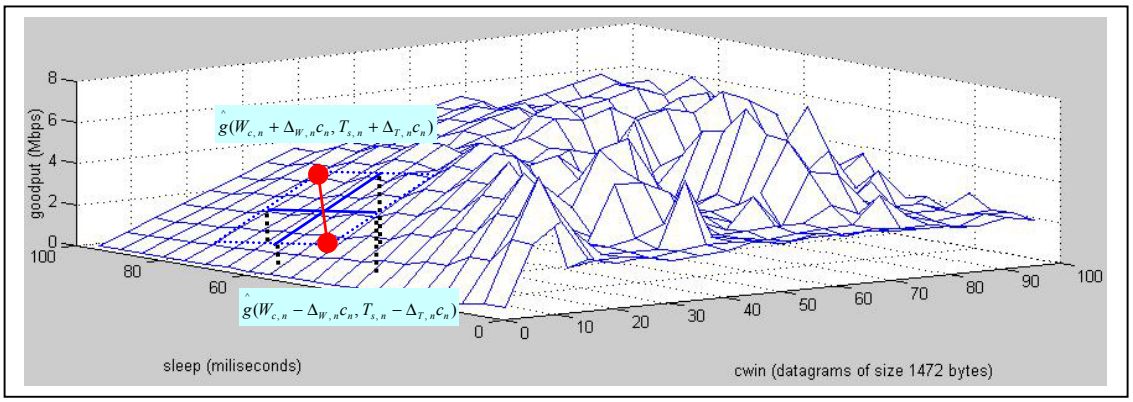


Figure 5.2 Gradient approximation in SPSA method

A one-measurement form of SPSA presented in [Spall97] requires only one function measurement independent of the number of parameter components being estimated. However, the further reduced number of measurements also brings the method one critical disadvantage of being more sensitive to small changes in the underlying data-generating process and to the choice of initial conditions. There exist some cases in which the standard SPSA method converges while the one-measurement SPSA might diverge. Although it is suggested that one of the most appropriate areas to apply the one-measurement SPSA method is in feedback control problems, we will not discuss its application any further in this dissertation work because the stochastic and complicated nature of the network traffic requires a robust transport control scheme.

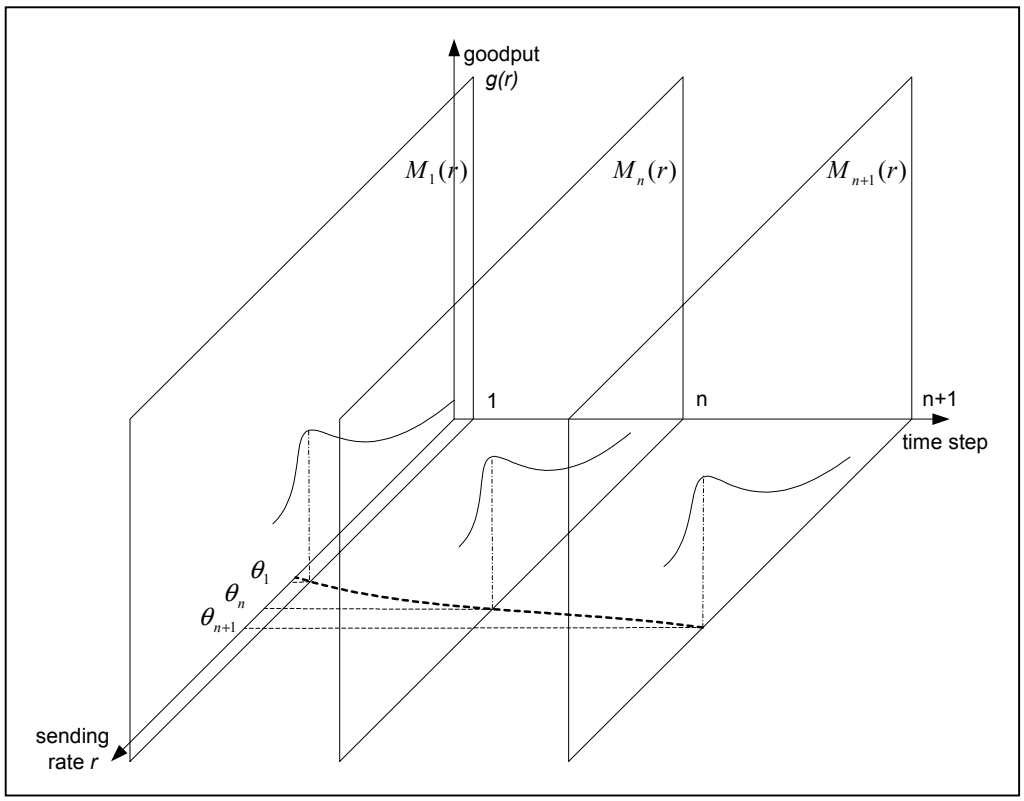


Figure 5.3 Time-varying goodput response in SPSA method

As discussed in Chapter 3, the goodput response regression function $M_n(r)$ may change shape indefinitely in the presence of dynamically changing background traffics, which cause the goodput maxima, i.e. the transition point between the two phases of the goodput response to change with time step *n*. Suppose that a sequence of real numbers $\theta_n, n = 1, 2, \cdots,$ are the true solutions of source rate *r* at which the goodput response regression function $M_n(r)$ achieve the unique maxima at time step *n*. We consider a time-varying version of the standard SPSA method for the goodput maximization problem. The time-varying goodput response in dynamic SPSA is illustrated in Figure 5.3.

We use the following recursive optimization procedure of the dynamic SPSA method to approximate the true solutions of the goodput maximization problem:

$$\begin{bmatrix} W_{c,n+1} \\ T_{s,n+1} \end{bmatrix} = \begin{bmatrix} W_{c,n} \\ T_{s,n} \end{bmatrix} + a_n \begin{bmatrix} \hat{G}_{W_{c,n}}(W_{c,n}, T_{s,n}) \\ \hat{G}_{T_{s,n}}(W_{c,n}, T_{s,n}) \end{bmatrix}$$

$$= \begin{bmatrix} W_{c,n} \\ T_{s,n} \end{bmatrix} + a_n \begin{bmatrix} \dfrac{\hat{g}(W_{c,n} + \Delta_{W,n}c_n, T_{s,n} + \Delta_{T,n}c_n) - \hat{g}(W_{c,n} - \Delta_{W,n}c_n, T_{s,n} - \Delta_{T,n}c_n)}{2\Delta_{W,n}c_n} \\ \dfrac{\hat{g}(W_{c,n} + \Delta_{W,n}c_n, T_{s,n} + \Delta_{T,n}c_n) - \hat{g}(W_{c,n} - \Delta_{W,n}c_n, T_{s,n} - \Delta_{T,n}c_n)}{2\Delta_{T,n}c_n} \end{bmatrix}$$

$$= \begin{bmatrix} W_{c,n} \\ T_{s,n} \end{bmatrix} + \frac{\dfrac{a}{n^{\alpha}}}{\dfrac{c}{n^{\gamma}}} \begin{bmatrix} \dfrac{\hat{g}(W_{c,n} + \Delta_{W,n}c_n, T_{s,n} + \Delta_{T,n}c_n) - \hat{g}(W_{c,n} - \Delta_{W,n}c_n, T_{s,n} - \Delta_{T,n}c_n)}{\Delta_{W,n}} \\ \dfrac{\hat{g}(W_{c,n} + \Delta_{W,n}c_n, T_{s,n} + \Delta_{T,n}c_n) - \hat{g}(W_{c,n} - \Delta_{W,n}c_n, T_{s,n} - \Delta_{T,n}c_n)}{\Delta_{T,n}} \end{bmatrix} \quad (5.5)$$

where $a > 0$ and $c > 0$ are two appropriate scalar gain coefficients, exponent $\frac{3}{5} < \alpha < 1$, and $\frac{\alpha}{6} \le \gamma < \alpha - \frac{1}{2}$.

The most interesting thing about the solution to the goodput maximization problem based on the dynamic SPSA method is its significance on fairness. If more transport sessions based on other protocols participate in sharing the common link, there is a larger chance of packet loss so that the goodput regression curve moves toward the coordinate origin and hence the optimal source rate decreases accordingly. On the contrary, the reduction in the background traffics makes the goodput regression curve move in the opposite direction and hence increases the optimal source rate. An extreme example is shown in Plot (b) of Figure 3.3, where the network is almost empty so that the optimal source rate nearly approaches the path bandwidth.

As seen from the statistical analysis of network traffic conducted in Chapter 3, without retransmission the maximum goodput is always achieved at an optimal source rate with very few packet losses. Although a higher throughput could be achieved even in the second phase of goodput response if a persistently aggressive retransmission strategy is

used, by no means we want the source rate to go beyond the optimal point for the sake of fairness. With appropriately selected coefficients, the dynamic SPSA method operates around the transition point between two phases of the goodput response and finally converges to the optimal source rate. This nice fairness property of the dynamic SPSA method is later justified by our experimental results.

5.4 Convergence of Dynamic SPSA

According to Equation (3.1), we are able to convert two control parameters, the pair of congestion window and sleep time to one single variable of source rate. Therefore, Equation (5.5) can be rewritten as:

$$r_{n+1} = r_n + \frac{a_n}{c_n}\left(\hat{g}_n^{+} - \hat{g}_n^{-}\right) = r_n + \frac{\frac{a}{n^{\alpha}}}{\frac{c}{n^{\gamma}}}\left(\hat{g}_n^{+} - \hat{g}_n^{-}\right) \tag{5.6}$$

where $\hat{g}_n^{+}$ and $\hat{g}_n^{-}$ are measurements of two conditionally independent random variables g_n^{+} and g_n^{-}, respectively, whose conditional expectations are the goodput response regression function values at the positively perturbed source rate and negatively perturbed source rate:

$$\begin{aligned} E[g_n^{+} \mid r_i, g_i, i < n, r_n] &= M_n(r_n + c_n) \\ E[g_n^{-} \mid r_i, g_i, i < n, r_n] &= M_n(r_n - c_n) \end{aligned} \tag{5.7}$$

Each conditional variance is bounded by a constant σ^2:

$$\begin{aligned} Var[g_n^{+} \mid r_i, g_i, r_n, i < n] &\le \sigma^2 \\ Var[g_n^{-} \mid r_i, g_i, r_n, i < n] &\le \sigma^2 \end{aligned} \tag{5.8}$$

Again, we assume that the true solutions $\theta_n, n = 1, 2, \cdots,$ to the dynamic maximum goodput problem vary in such a way that

$$\theta_{n+1} - \theta_n = O(n^{-\omega}) \tag{5.9}$$

where $\omega > \alpha$. We further assume that the initial goodput response regression $M_1(r)$ is increasing for $r < \theta_1$ and decreasing for $r > \theta_1$. There exist three real-valued constants K_0, K_1, and K_2 such that the first and third derivatives of $M_1(r)$ are confined by:

$$K_0 \mid r - \theta_1 \mid \le \mid M_1^{'}(r) \mid \le K_1 \mid r - \theta_1 \mid \tag{5.10}$$

$$\mid M_1^{'''}(r) \mid \le K_2 \qquad \text{for all } -\infty < r < +\infty \tag{5.11}$$

Same as in dynamic RMSA, the expectation of the square of the initial source rate r_1 is constrained by

$$E[r_1^2] < +\infty \tag{5.12}$$

Theorem 5.1: Under the assumptions (5.7), (5.8), (5.9), (5.10), (5.11), and (5.12), the approach to the goodput maximization problem defined by the recursive optimization procedure in Equation (5.6) converges to a true solution θ_n in the quadratic mean:

$$E[(r_n-\theta_n)^2]=\begin{cases} O(n^{-(\alpha-2\gamma)}) & \text{for} \quad \omega \geq \frac{3}{2}\alpha-\gamma \\ O(n^{-[2(\omega-\alpha)]}) & \text{for} \quad \omega < \frac{3}{2}\alpha-\gamma \end{cases} \tag{5.13}$$

Proof of Theorem 5.1:

Similar to the discussion we made on the assumptions for the convergence proof of TCOU for goodput stabilization, the convergence conditions (5.7), (5.8), (5.9), (5.10), (5.11), and (5.12) are reasonable assumptions under normal network traffic conditions according to the extensive network performance measurements we collected in Chapter 3. The formal convergence proof mostly follows the one given in [Dupac65] except that our solution is based on an unmodified dynamic SPSA method. We present it here anyway for the sake of completeness.

As stated earlier, real numbers $\theta_n, n=1,2,3,\cdots$, are true solutions to the maximum goodput problem: $\underset{r_n}{Max}: M_n(r_n)$. Again, we set $M_1(r)=M(r)$ and $M_n(r)=M(x-\theta_n+\theta_1)$. The finite difference of the initial goodput regression at source rate *r* is calculated as

$$dM(r)=\frac{M(r+c)-M(r-c)}{c} \tag{5.14}$$

Extend the numerator of Equation (5.14) at point *r* in Taylor's Series and use condition (5.10), (5.11):

$$\begin{aligned} & dM(r)=\frac{1}{c}\big(M(r+c)-M(r-c)\big) \\ & \frac{1}{c}\left(M(r)+\frac{M^{'}(r)}{1!}(r+c-r)+\frac{M^{''}(r)}{2!}(r+c-r)^2+\frac{M^{'''}(r)}{3!}(r+c-r)^3+O(c^4)\right)- \\ & \frac{1}{c}\left(M(r)+\frac{M^{'}(r)}{1!}(r-c-r)+\frac{M^{''}(r)}{2!}(r-c-r)^2+\frac{M^{'''}(r)}{3!}(r-c-r)^3+O(c^4)\right) \\ & \approx \frac{1}{c}\left(2M^{'}(r)c+\frac{1}{3}M^{'''}(r)c^3\right) \\ & =2M^{'}(r)+\frac{1}{3}M^{'''}(r)c^2 \\ & =-\mu(r-\theta_1)+\lambda c^2 \end{aligned} \tag{5.15}$$

where $2K_0 \leq \mu \leq 2K_1$ and $|\lambda| \leq \frac{1}{3}K_2$.

Accordingly, the finite difference of goodput regression at time step *n* has the following expression:

$$dM_n(r)=dM(r-\theta_n+\theta_1)=-\mu(r-\theta_n)+\lambda c_n^{\ 2} \tag{5.16}$$

which is actually the conditional expectation of difference between random variables g_n^{+} and g_n^{-} divided by the difference interval:

$$E[(\frac{g_n^{+} - g_n^{-}}{c_n}) \mid r_i, g_i, i < n, r_n] = dM_n(r) = -\mu(r - \theta_n) + \lambda {c_n}^2 = -\mu(r - \theta_n) + \lambda \frac{c^2}{n^{2\gamma}} \quad (5.17)$$

From the property of the variance of a random variable X: $Var[X] = E[X^2] - (E[X])^2$, it follows that $E[X^2] = Var[X] + (E[X])^2$. Using Equation (5.16), (5.33), and (5.35), we obtain an upper bound for the conditional expectation of $(g_n - g^*)^2$:

$$\begin{aligned} & E[(\frac{g_n^{+} - g_n^{-}}{c_n})^2 \mid r_i, g_i, r_n, i < n] \\ & = Var[(\frac{g_n^{+} - g_n^{-}}{c_n}) \mid r_i, g_i, r_n, i < n] + (E[(\frac{g_n^{+} - g_n^{-}}{c_n}) \mid r_i, g_i, r_n, i < n])^2 \\ & = 2\frac{\sigma^2}{{c_n}^2} + [dM_n(r)]^2 \\ & \le K_3 n^{2\gamma} + K_4 (r_n - \theta_n)^2 \end{aligned} \quad (5.18)$$

where K_3 and K_4 are appropriate coefficients.

Subtract both sides of Equation (5.9) from both sides of Equation (5.6) and square:

$$\begin{aligned} (r_{n+1} - \theta_{n+1})^2 &= [r_n - \theta_n + \frac{\frac{a}{n^\alpha}}{\frac{c}{n^\gamma}}(g_n^{+} - g_n^{-}) + O(n^{-\omega})]^2 \\ &= (r_n - \theta_n)^2 + \frac{a^2}{n^{2\alpha}}\left(\frac{g_n^{+} - g_n^{-}}{c_n}\right)^2 + O(n^{-2\omega}) \\ &\quad + 2(r_n - \theta_n)\frac{a}{n^\alpha}\left(\frac{g_n^{+} - g_n^{-}}{c_n}\right) + 2(r_n - \theta_n)O(n^{-\omega}) + 2\frac{a}{n^\alpha}\left(\frac{g_n^{+} - g_n^{-}}{c_n}\right)O(n^{-\omega}) \end{aligned} \quad (5.19)$$

Take conditional expectation of Equation (5.19), and use Equation (5.9), (5.17), and (5.18):

$$\begin{aligned} & E[(r_{n+1} - \theta_{n+1})^2 \mid r_i, g_i, i < n, r_n] = (r_n - \theta_n)^2 + \frac{a^2}{n^{2\alpha}}(K_3 n^{2\gamma} + K_4 (r_n - \theta_n)^2) + O(n^{-2\omega}) \\ & + 2 \mid r_n - \theta_n \mid O(n^{-\omega}) + 2(r_n - \theta_n)\frac{a}{n^\alpha}\left(\frac{g_n^{+} - g_n^{-}}{c_n}\right) + 2\frac{a}{n^\alpha}\left(\frac{g_n^{+} - g_n^{-}}{c_n}\right)O(n^{-\omega}) \\ & \le \frac{K_5}{n^{2\alpha - 2\gamma}} + (1 - \frac{K_6}{n^{2\alpha}})(r_n - \theta_n)^2 + \frac{K_7}{n^\omega} \mid r_n - \theta_n \mid - \frac{K_8}{n^\alpha}(r_n - \theta_n)^2 + \frac{K_9}{n^{\alpha + 2\gamma}} \mid r_n - \theta_n \mid \\ & \le \frac{K_5}{n^{2\alpha - 2\gamma}} + (1 - \frac{K_8}{n^\alpha})(r_n - \theta_n)^2 + \frac{K_7}{n^\omega} \mid r_n - \theta_n \mid + \frac{K_9}{n^{\alpha + 2\gamma}} \mid r_n - \theta_n \mid \end{aligned} \quad (5.20)$$

where K_5, K_6, K_7, K_8, and K_9 are appropriate coefficients.

Again take unconditional expectation of Equation (5.20) and use inequality $E[|X|] \le \varepsilon + \frac{1}{\varepsilon} E[X^2]$ to estimate the last two terms in Equation (5.20) with $\varepsilon = \frac{1}{\delta n^{\omega-\alpha}}$ and $\varepsilon = \frac{1}{\delta' n^{2\gamma}}$, respectively:

$$\begin{aligned} E[(r_{n+1} - \theta_{n+1})^2] &\le \frac{K_5}{n^{2\alpha-2\gamma}} + (1 - \frac{K_8}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_7}{n^\omega} \frac{1}{\delta n^{\omega-\alpha}} + \frac{K_9}{n^{\alpha+2\gamma}} \frac{1}{\delta' n^{2\gamma}} \\ &= \frac{K_5}{n^{2\alpha-2\gamma}} + (1 - \frac{K_8}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_{10}}{n^{2\omega-\alpha}} + \frac{K_{11}}{n^{\alpha+4\gamma}} \end{aligned} \quad (5.21)$$

where K_{10} and K_{11} are appropriate coefficients. Since $\frac{\alpha}{6} \le \gamma < \alpha - \frac{1}{2}$, we have $\alpha \le 6\gamma$, which in turn implies that $2\alpha - 2\gamma \le \alpha + 4\gamma$, the last two terms in Equation (5.21) can be combined:

$$E[(r_{n+1} - \theta_{n+1})^2] \le \frac{K_5}{n^{2\alpha-2\gamma}} + (1 - \frac{K_8}{n^\alpha}) E[(r_n - \theta_n)^2] + \frac{K_{12}}{n^{\alpha+4\gamma}} \quad (5.22)$$

where K_{12} is an appropriate coefficient. The application of Chung's Lemma 4, i.e. Equation (4.42) and Equation (4.43), completes the proof.

5.5 Throughput Maximization Without Congestion Control

The goodput and loss rate curves in response to source rate without retransmission plotted in Figure 3.3 shows that in the congestion collapse phase, a fast source rate beyond a certain threshold value incurs an irregular high loss rate, which may result in a low throughput. However, with an extremely high source rate as well as an aggressive retransmission strategy, a higher throughput could be achieved even if the source rate is operated in the second phase of the goodput response curve. In this section, we discuss non-congestion-controlled file transfer protocols, such as SABUL, Tsunami, and Hurricane.

SABUL was developed by researchers at University of Illinois at Chicago. It is a reliable UDP-based protocol intended for large data transfers over high-speed wide area networks [Sabul]. A SABUL sender uses UDP channel to send large amounts of data to a SABUL receiver, which feeds back the state information of the packet loss and the list of lost packets to the sender using TCP channel. Based on the state information obtained from the receiver, the sender dynamically adjusts its sending rate to minimize the packet loss.

Tsunami was developed by researchers at Indiana University's Advanced Network Management Lab, physicists at TRIUMF, Canada's national laboratory for particle and nuclear physics in 2002. Tsunami is designed to transfer very large data files over great distances. From the source code released by IU, we found that Tsunami has a similar architecture of data transfer and rate control as SABUL. The optimal source rate in Tsunami is targeted at 1Gbps. The developers claimed that a file transfer speed as high as 769 Mbps has been achieved [Tsunami].

For comparison purpose, we developed our version of aggressive file transfer protocol, named Hurricane, which is directly modified from our UDP and TCP based network measurement program used in Chapter 3. This simple file transfer protocol has a very loose rate control so that the Hurricane sender persistently maintains a high sending rate

even in the case of high loss rate. As illustrated in Figure 5.4, the architecture of Hurricane is very similar to those in SABUL and Tsunami.

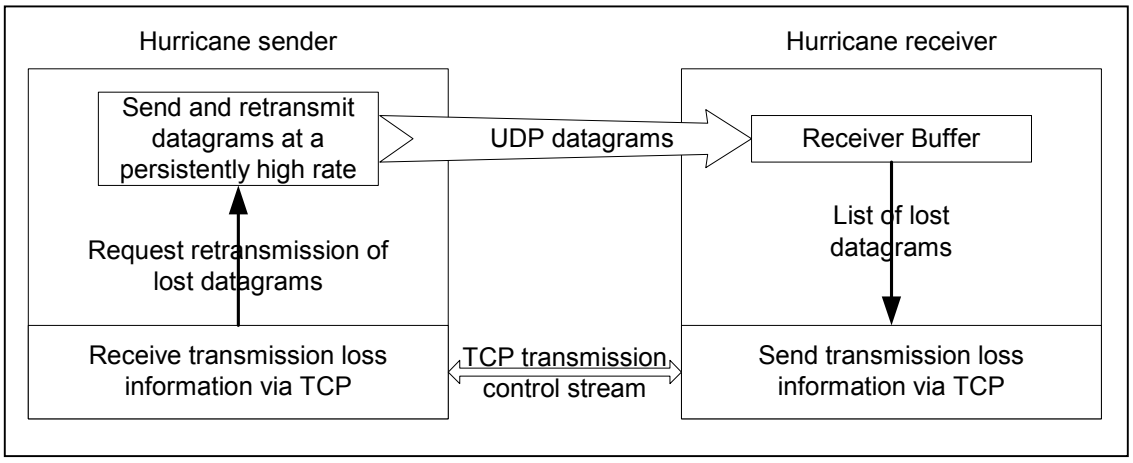


Figure 5.4 Overview of Hurricane structure

We run Tsunami and Hurricane on different networks individually and collect the corresponding performance measurements as references to the experimental results from TCOU for goodput maximization. The measurements of Tsunami and Hurricane made on two links with different link characteristics are listed in Table 5.1.

Table 5.1 Performance measurements for Tsunami and Hurricane

Protocols	Performance Measurements	From: ozy4.csm.ornl.gov To: resource.rrl.lsu.edu Message size: 50M bytes Datagram size: 1472 bytes Bandwidth: 10.0 Mbps	From: firebird.ccs.ornl.gov To: ccil.cc.gatech.edu Message size: 200M bytes Datagram size: 32768 bytes Bandwidth: 1.0 Gbps
Tsunami	Sending rate	9.51 Mbps	319.77 Mbps
	Throughput	9.0 Mbps	108.98 Mbps
	Loss rate	5.05%	60.57%
Hurricane	Sending rate	9.32 Mbps	135.67 Mbps
	Throughput	9.11 Mbps	102.97 Mbps
	Loss rate	2.26%	24.32%

As shown in Table 5.1, an appropriate datagram size of 1472 bytes (i.e. MTU – UDP header – IP header) is selected for the link between ozy4.csm.ornl.gov and resource.rrl.lsu.edu because this link only has a path MTU of 1500 bytes. The NIC speed of ozy4.csm.ornl.gov and resource.rrl.lsu.edu as well as the path bandwidth between them is 10.0 Mbps, while the NIC speed of firebird.ccs.ornl.gov and ccil.cc.gatech.edu as well as the path bandwidth between them is 1.0 Gbps.

For the link between ozy4.csm.ornl.gov and resource.rrl.lsu.edu, Tsunami and Hurricane send datagrams at a rate close to the bandwidth. High throughput and bandwidth

utilization is achieved in both cases with a reasonably low loss rate. However, all participating TCP sessions are killed because TCP's AIMD algorithm keeps backing off congestion window so that most of the link resources are occupied by Tsunami or Hurricane. For the link between firebird.ccs.ornl.gov and ccil.cc.gatech.edu, both Tsunami and Hurricane experience high loss rate but still achieve comparably high throughput. Again, their high sending rate and aggressive retransmission scheme suppress all other coexisting TCP sessions.

It is not desirable to achieve high individual throughput or bandwidth utilization by unfairly grabbing a large fraction of the available bandwidth, especially in times of network congestion. [FF99] discusses the negative impacts such as unfairness and congestion collapse of the deployment of non-congestion-controlled traffic in the Internet, which is classified into three categories: "Not TCP-friendly", unresponsive, and using disproportionate bandwidth. The identification method as well as its bandwidth share restriction technique employed in routers is proposed for each type of traffic not using end-to-end congestion control.

The network resources in packet-switching networks are meant to be fairly shared by all concurrent data flows. Therefore, it is of our particular interest to examine the fairness problem in TCOU for goodput maximization. Since TCOU has congestion control based on the dynamic SPSA method, we expect it to excel Tsunami and Hurricane in this regard.

5.6 Implementation and Experimental Results

TCOU for throughput maximization is built on the same fundamental framework discussed in Chapter 4, which involves all techniques to guarantee transmission reliability such as datagram sequencing and acknowledging, three-way handshake and connection termination, floating window based flow control, packet loss detection and lost packet retransmission, etc.

5.6.1 Framework of TCOU for Goodput Maximization

Same as in the goodput stabilization, a source node conducts all the rate control related activities, while a destination node does nothing but receives and acknowledges each arriving datagram. The TCOU receiver has the exactly same structure as the one for goodput stabilization described in Chapter 4. Therefore, we only present the control flow diagram of TCOU sender for throughput maximization in Figure 5.5.

As Figure 5.5 shows, the TCOU sender for goodput maximization also consists of two child threads, one of which, the datagram sending thread, provides the same functionality as its counterpart in goodput stabilization. The acknowledgement receiving thread alternates positive and negative perturbation and calculates the corresponding goodput and loss rate, based on which, the dynamic SPSA is performed on the *cwin* and sleep time simultaneously to control the source rate. The perturbation duration is determined by the estimated RTT and is always set long enough to make the perturbation take effect.

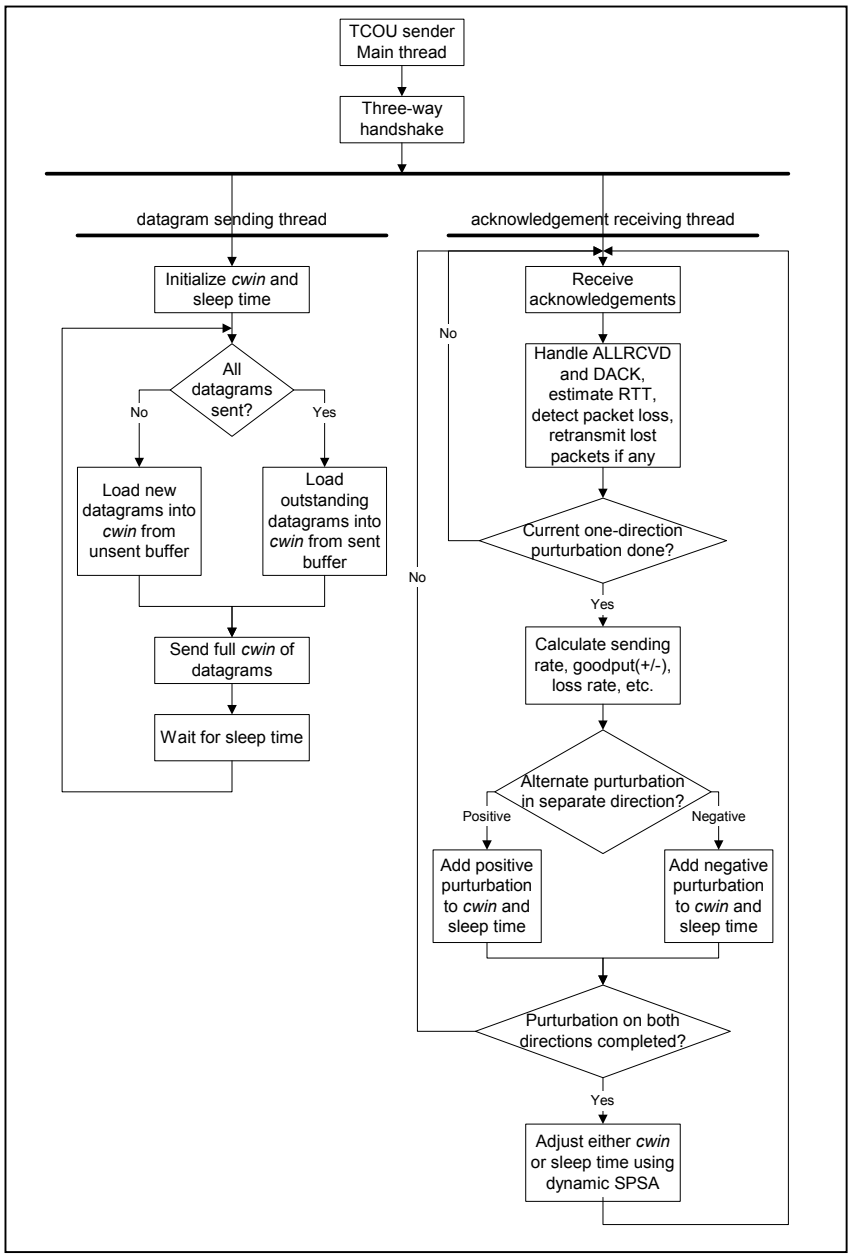


Figure 5.5 Control flow diagram of TCOU sender for goodput maximization

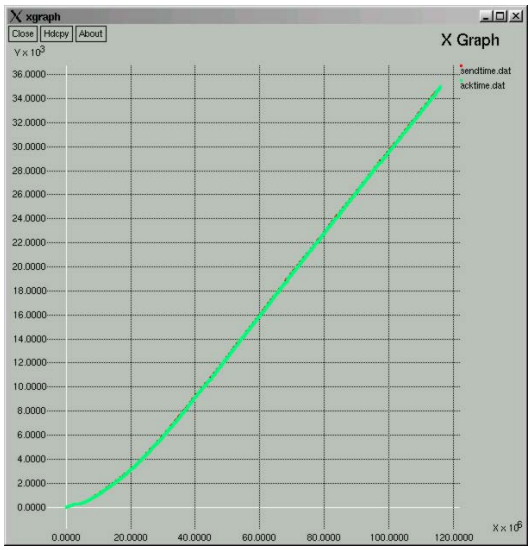
5.6.2 Experimental Results

The throughput maximization experiments are conducted between the host ozy4.csm.ornl.gov at ORNL and the host resource.rrl.lsu.edu at LSU. The TCOU sender creates a test message of 50M bytes and sends it to the TCOU receiver as a set of UDP datagrams, each of which is of 1472 bytes except the last one. To investigate the robustness and fairness of our method, we run on-host and off-host regular background network traffics during the experiments.

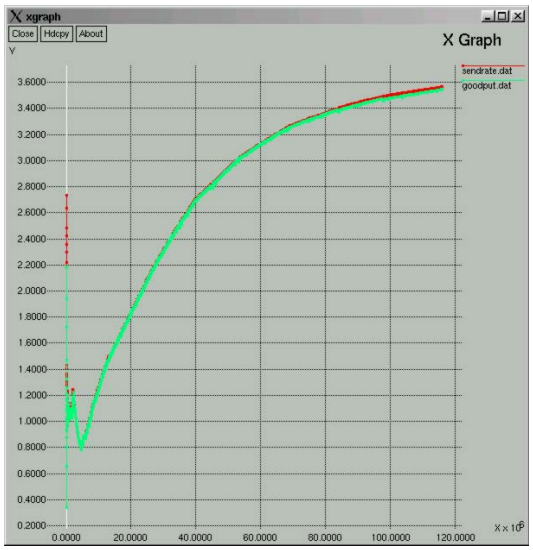
Case 1: Low starting point

In this case, TCOU sender starts with a low sending rate: *cwin* = 8 datagrams and sleep time = 90 ms. Coefficients of the dynamic SPSA method are chosen as follows: a =

800.0, $\alpha = 0.7$, c = 10.0, $\gamma = \dfrac{(\alpha/6.0 + \alpha - 0.5)}{2.0}$. Same as other cases where the stochastic approximation methods are applied, there are no strict rules for selecting coefficient values except satisfying those loose convergence conditions. In most practical applications, these coefficient values are empirically determined. We choose such values that the step size adjustment made on *cwin* and sleep time is valid or meaningful. Since there is a delay of RTT between the sending and acknowledging of datagrams, the perturbation imposed on *cwin* and sleep time does not come into effect immediately. We set the perturbation duration as five times RTT to wait for the perturbation to take effect.
The performance measurements are plotted in Figure 5.6. Plot (a) shows the curves of the datagram sending and acknowledging time vs. datagram sequence numbers, and Plot (b) shows the curves of the datagram acknowledging time vs. the source sending rate and goodput.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers



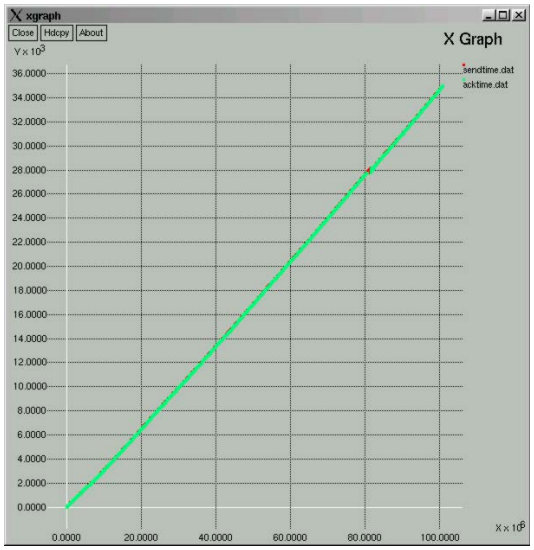
(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

Figure 5.6 Goodput maximization with low starting point

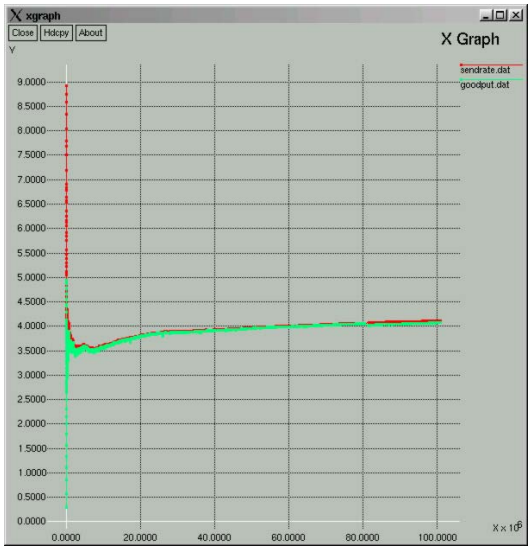
As Figure 5.6 shows, the sending rate is increased quickly in the beginning, so is the corresponding goodput due to its low initial position with very little packet loss. This climb-up tendency is gradually slowing down, especially when it approaches the target maximum goodput level. The whole evolution process of sending rate and goodput exhibits a nice smooth concave shape. TCOU achieves a goodput above 3.5Mbps with an overall loss rate 0.87%, and more importantly it does not have any significant impact on the performance of all other concurrent regular TCP sessions such as web browsing, email services, telnet, etc.

Case 2: High starting point

In this case, we select a high starting point with *cwin* = 30 datagram and sleep time = 100 ms for the TCOU sender while all other network settings remain the same. Figure 5.7 shows the performance measurements.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers

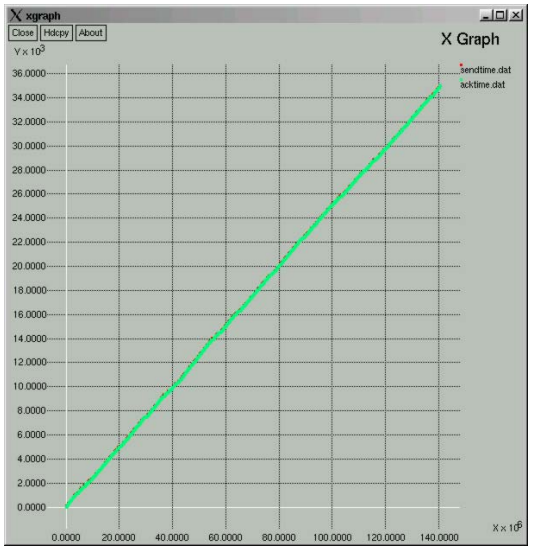


(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

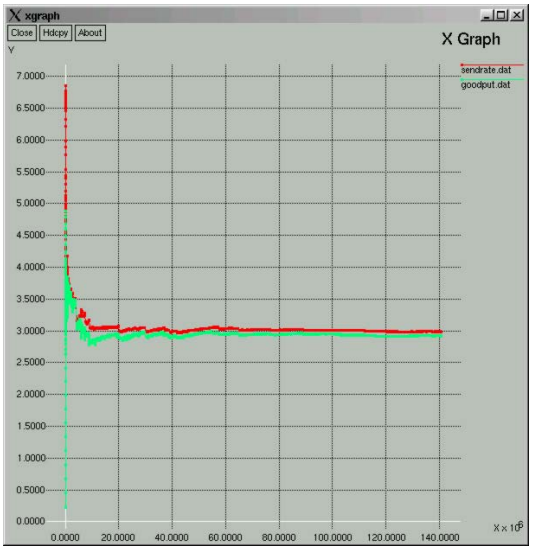
Figure 5.7 Goodput maximization with high starting point

Compare Plot (b) in Figure 5.6 and Figure 5.7. The goodput evolution process in this case also exhibits a concave curvature in an overall sense, but the goodput converges to the target maximum level much faster than in Case 1 with a low starting point. A steady goodput as high as larger than 4.0 Mbps is eventually achieved by TCOU with an overall loss rate 0.73%. Again, there is no perceivable impact on the performance of other coexisting regular TCP sessions.

Case 3: TCOU competing with large file transfer using FTP



(a) Datagram sending and acknowledging time (μs) vs. datagram sequence numbers



(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

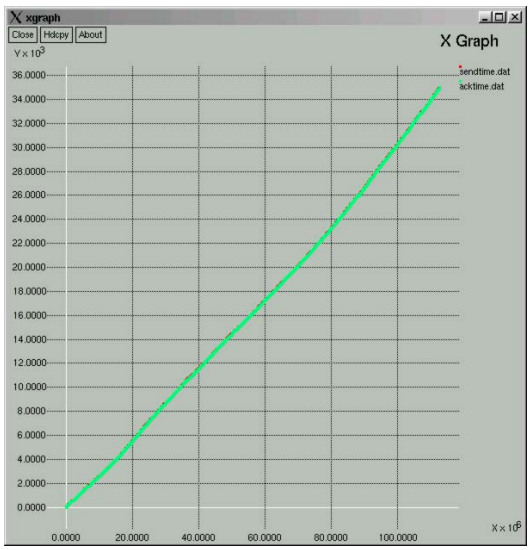
Figure 5.8 TCOU for maximization competing with large file transfer using FTP

To further explore the fairness property of TCOU for goodput maximization, besides regular background traffics, we particularly use FTP to transfer a large file through the same link in the same direction during the whole period of TCOU session. All the network settings, coefficient values, and initial position of TCOU sender remain the same as those in Case 2. The performance measurements are plotted in Figure 5.8.

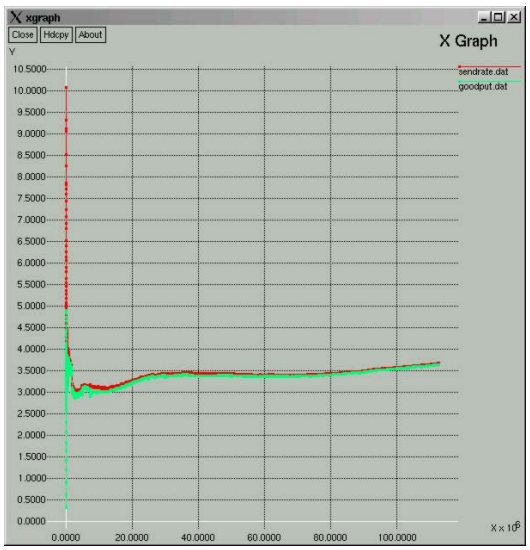
In this case, TCOU achieves a steady goodput of 2.95 Mbps with a loss rate of 2.09%, and TCP achieves about 1.0 Mbps, which is a typical goodput level under normal traffic condition. The sum of goodputs achieved by both TCOU and TCP is approximately equal to the one achieved by the single TCOU in the previous case. Obviously, TCOU yields about 1.0 Mbps of bandwidth share to the competing FTP session. In other words, TCOU makes most use of the available bandwidth without stealing any share from the concurrent TCP flow. Another observation worth being pointed out is that the high starting point is lower than the target source rate in Case 2 and higher than the target source rate in Case 3. The goodput of TCOU in both cases eventually converges to its own maximum goodput level.

Case 4: TCOU with different coefficient values

In this case, we try to investigate how the coefficient values in the dynamic SPSA method affect the network performance. A different set of values are chosen for the coefficients as follows: a = 1000.0, $\alpha = 0.65$, c = 8.0, $\gamma = \frac{(\alpha/6.0 + \alpha - 0.5)}{2.0}$, and the performance measurements are shown in Figure 5.9.



(a) Datagram sending & acknowledging time (μs) vs. datagram sequence numbers



(b) Datagram acknowledging time (μs) vs. source rate (Mbps) & goodput (Mbps)

Figure 5.9 TCOU for maximization with different coefficient values

TCOU achieves an overall goodput of 3.7 Mbps with 0.73% loss rate. Also we observed that the goodput evolution process in this case is not as smooth as those in Case 2 and Case 3. This phenomenon can be explained by the fact that the increase of coefficient a and decrease of coefficients α and c generates a bigger step adjustment size, which causes more oscillations in the optimization process.

For comparison, we list in Table 5.2 the average performance measurements of Tsunami, TCOU for goodput maximization, and TCP on the same link between ozy4.csm.ornl.gov and resource.rrl.lsu.edu.

Table 5.2 Performance comparisons between Tsunami, TCOU, and TCP

Protocols	Message size (Mbytes)	Datagram size (bytes)	Average goodput (Mbps)	Overall loss rate (%)	Killing other TCP
Tsunami	50	1472	9.0	5.0	Yes
TCOU	50	1472	4.2	0.7	No
TCP	50	1460	< 1.0	N/A	No

Based on the experiments conducted above, we have the following conclusions:

(1) In the current implementation of TCOU, we only use goodput measurements to guide the optimum search process. In some case where TCOU runs with aggressive protocols like Tsunami, TCOU may not perform as well as expected if it keeps trapped in the second phase of the goodput response. In our future work, datagram loss rate may be considered to help TCOU jump out of the second phase at the early stage.

(2) Same as TCOU for goodput stabilization, the selection of starting point has impact on the convergence speed. An appropriate initial source rate can be determined based on some bandwidth measurement schemes.

(3) Different coefficient values in the dynamic SPSA method also affect the source rate optimization process, but not as significantly as in the dynamic RMSA method.

(4) Many aspects of the fundamental structure of TCOU can be further improved. For example, instead of acknowledging every single datagram, we may apply a delayed acknowledgment scheme as in TCP.

CHAPTER 6 OVERLAY NETWORK OF NETLETS

6.1 Introduction

Overlay Network (ON) is the fundamental building block of our proposed ONTCOU. In this chapter, we discuss all developmental and implemental issues in some detail related to overlay network based on NetLets daemons. Designing overlay network and building TCOU on it have been motivated by the following considerations.

Firstly, like most other protocols for end-to-end transport control, TCOU itself lacks the ability of choosing efficient routes for data transmission. However, as it is well known, routing plays a significantly important role in determining the end-to-end performance in the distributed computing applications in the Internet. Generally, a default routing path is determined by intermediate routers based on the best-effort mechanism with the intention of minimizing the number of hops from source to destination. Obviously, a path with the minimum number of hops is not necessarily the best one because in addition to the number of hops, available path bandwidth and network congestion condition are also vital factors in deciding the end-to-end delay. To ensure end-to-end performance, we design an overlay network of NetLets daemons, which perform extensive active network traffic measurements, estimate available path bandwidth, compute multiple quickest paths, and conduct data routing at the application level.

Secondly, since the transport control layer is conventionally integrated in Operating Systems, deploying any new algorithms and protocols for transport control usually requires enormous changes on the existing network infrastructures. Unfortunately, such changes are very unlikely to happen in consideration of the colossal size of the current Internet. An overlay network is designed to reside at the application level so that its deployment requires no changes to the existing Internet infrastructures. Therefore, the overlay network turns out to be an appropriate place to implement our new transport control protocol TCOU. As a matter of fact, a great number of applications have used overlay networks to deploy or test new protocols and services in order to have a minimal impact on the lower IP infrastructures [JGJKO00, ABKM01, TH98]. Meanwhile, we are also aware of all the computing overheads that may arise in the overlay network implemented at the application level.

6.2 Measurement based Bandwidth Estimation

With the increasing growth and wide use of the Internet, more and more demands are being placed on the network performance. The accurate and extensive performance measurement, especially the bandwidth estimation is required to achieve high quality of service from the currently available network resources.

The link bandwidth is the fastest rate at which the signal of bits can be generated and inserted into the physical medium and the available link bandwidth is the spare bandwidth of the link “left over” after the cross traffic. Obviously, the link with the minimum link bandwidth on the path may not be the one with the lowest available link bandwidth. A data path usually consists of multiple physical links and the path bandwidth is defined by the minimum of the link bandwidths on the path. The available path bandwidth is the maximum throughput the path can provide to a flow given the current cross traffic load on the path. Recently a great deal of work has been conducted on the

measurement of the link and path bandwidth [CM01, DRM02] as well as the available bandwidth [JD02]. Our method based on active measurement is to estimate the available path bandwidth.

As we discussed earlier in Chapter 1, there are three main types of delays involved in the data transmission over computer networks: link propagation delay d_p, equipment-associated delay (mostly due to queuing delay) d_q, and bandwidth-constrained delay d_{BW}. Due to dynamic changes in the network condition, the queuing delay usually experiences a high level of randomness and hence is the most complicated delay component. We have the following expression for the end-to-end delay of transmitting a message of size *r* on a path *P* with *l* links or hops:

$$d(P,r) = d_{BW}(P,r) + \sum_{i=1}^{l} (d_{p,i}(P) + d_{q,i}(P,r)) \tag{6.1}$$

If the message size *r* is much smaller than the path MTU, the smallest MTU on the path between two end nodes, and the network is lightly loaded, the bandwidth-constrained delay in Equation (6.1) is negligible so that the end-to-end delay mostly accounts for the sum of the queuing and path propagation delay. We may denote this lower boundary of the end-to-end delay by a fixed quantity $d_{\min}(P) = \sum_{i=1}^{l} (d_{p,i}(P) + d_{q,i}(P,r)), \quad r << MTU$.

On the other hand, if the message size *r* is very large, the end-to-end delay is mostly determined by the bandwidth-constrained delay perturbed by a small quantity of "noise" imposed by queuing delay. Let *ABW*(*P*) denote the available path bandwidth. The end-to-end delay $d(P,r)$ for transmitting a message of size *r* can be approximately linearized as:

$$d(P,r) \approx \frac{1}{ABW(P)} r + d_{\min}(P) \tag{6.2}$$

In a circuit-switching network like the telephone system, the maximum transmission rate is fixed and determined by the minimum available link bandwidth *ABW*(*link*) of the path *P* between two endpoints:

$$ABW(P) = \min_{link \in P} \{ABW(link)\} \tag{6.3}$$

In a packet-switching network like the computer network, all data packets are stored and forwarded at intermediate nodes. Thus the bandwidth-constrained delay needs to be counted at every component link. For the transmission of a packet of length less than the path MTU, the available path bandwidth is approximated in [RGBR00] as:

$$ABW(P) = \frac{1}{\sum_{link \in P} \frac{1}{ABW(link)}} \tag{6.4}$$

However, for the transmission of a message of large size, the pipelining of data packets on component links makes the analytical calculation of the available path bandwidth be practically close to Equation (6.3).

Estimation of available path bandwidth is a crucial part of overlay network because it provides important information about resource utilization and congestion condition for other function units of NetLets daemon. Here we particularly point out that a virtual link in our proposed overlay network corresponds to a physical path in the underlying

network, which usually consists of multiple actual links. The bandwidth of a virtual link or a physical path is estimated through active measurements based on the following steps:

Step 1. The source node generates a set of messages with various sizes *s*.

Step 2. The source node divides each size of message into a number of message components of a certain read/send line length and transmits them to the destination node through a TCP channel. Note that internally each message component is chunked into segments of MSS at the TCP layer, each of which is probably further fragmented into data packets at the IP layer, depending on the underlying link MTU.

Step 3. The destination node receives each message component and immediately echoes it back to the source node.

Step 4. The source node receives each message component fed back by the destination and accumulates the two-way transmission duration until all message components are received for a certain size of message being transmitted. This whole process is repeated a couple of times for each message size to compute the average round-trip time. The end-to-end message transmission delay d is estimated as the round-trip time of message transmission divided by two.

Step 5. Once the average end-to-end delays are determined for all messages of different sizes, we apply a linear regression to fit the measured points of message size r and end-to-end delay d pair. The first order approximate of the available path bandwidth ABW and the minimum end-to-end delay $d_{\min}$ are then estimated by the slope and intercept of the regression line $d = \frac{1}{ABW} r + d_{\min}$, respectively.

```
Generate a set of messages with various sizes;
Divide each message into a number of message components of a certain line length;
For each size of message
{
        For each times of measurement
        {
                Do
                {
                        Read and send a message component;
                        Receive echoed message component;
                        Accumulate two-way transmission duration;
                }
                Until all message components are sent and received
        }
        Compute average round-trip time;
        Estimate end-to-end delay for the current size of message;
}
Apply a linear regression analysis on all points of message size and end-to-end delay pair;
Estimate bandwidth and end-to-end delay using slope and intercept of the regression line.
```

Figure 6.1 Algorithm used for bandwidth estimation at source node in NetLets

As we see above, the destination node does nothing but echo back message components immediately after it receives them. Most activities like end-to-end delay measurement

and linear regression analysis are performed at the source node. We briefly describe the algorithm used for bandwidth estimation at the source node in Figure 6.1.

The end-to-end message transmission delay measurements between two NetLets deployed at LSU and ORNL as well as its corresponding linear regression estimate are illustrated in Figure 6.2. The measurement of the round-trip transmission delay for each message size is carried out 3 times and the end-to-end delay plotted in the figure is the half of the average value.

From this computation, we estimate the available path bandwidth ABW of this virtual link as about 1.0 Mbps. Since the minimum end-to-end delay is estimated from the intercept of the regression line, which is more sensitive to the measurements, we may not be able to obtain a valid estimation of the minimum end-to-end delay in some case where there exists unstable network conditions or inaccurate measurements. An alternative way to estimate the minimum end-to-end delay is to send a message containing just one byte for a certain number of times and average the round-trip time measurements. However, the actual frame size delivered on the physical medium between two end nodes is more than 41 bytes on account of the TCP and IP headers as well as the link layer header and trailer.

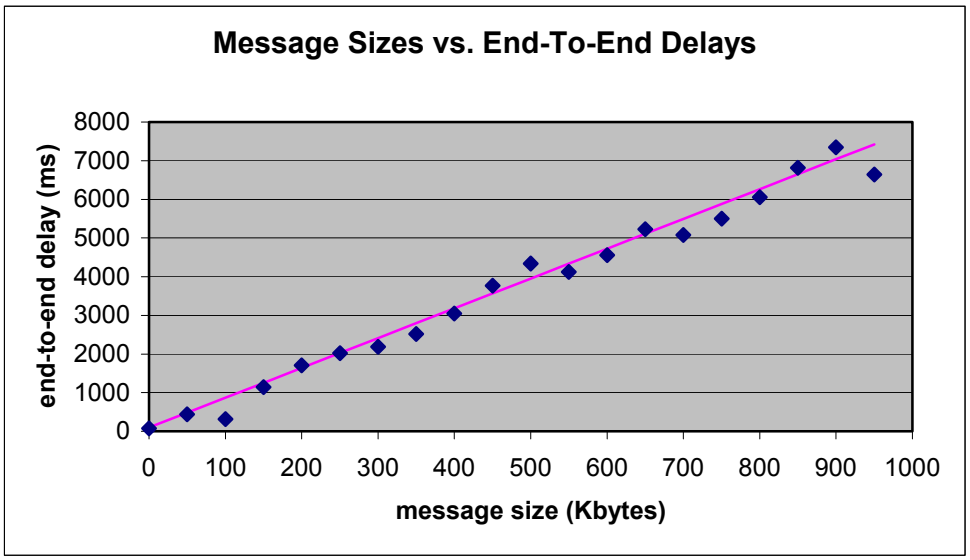


Figure 6.2 End-to-end delay measurements for messages of various sizes transmitted between LSU and ORNL

We use the following theorem to perform the linear regression estimate for the available path bandwidth and the minimum end-to-end delay:

Theorem 6.1: Given a set of test messages with various sizes $R = \{r_i \mid i = 1,2,...,k\}$, the corresponding end-to-end delays (over one virtual or physical link) are measured as $D = \{d_i \mid i = 1,2,...,k\}$. The following formula gives the coefficient vector of a polynomial regression estimate in the Least Squares sense.

$$\vec{a} = (X^T X)^{-1}(X^T \vec{y}) \tag{6.5}$$

where, $\vec{a}$ is the coefficient vector of a polynomial regression estimate: $d = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + ... + a_1 r + a_0$. Column vector $\vec{y} = D$, and matrix X is constructed as follows:

$$X = \begin{bmatrix} r_1^{n-1} & r_1^{n-2} & \dots & r_1 & 1 \\ r_2^{n-1} & r_2^{n-2} & \dots & r_2 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ r_k^{n-1} & r_k^{n-2} & \dots & r_k & 1 \end{bmatrix} \tag{6.6}$$

Proof of Theorem 6.1: For the given observations, test message sizes R and measured end-to-end delays D, the coefficients of a polynomial regression estimate $d = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \dots + a_1 r + a_0$ are determined by the Least Squares method such that:

$$\varphi = \min\left(\sum_{i=1}^{k} \varepsilon_i^{\,2}\right) \tag{6.7}$$

where the residual $\varepsilon_i = a_{n-1}r_i^{n-1} + a_{n-2}r_i^{n-2} + \dots + a_1 r_i + a_0 - d_i, \quad (i = 1,2,\dots,k)$. By taking partial derivative with respect to the unknown coefficients, we have:

$$\begin{cases} \dfrac{\partial \varphi}{\partial a_0} = \sum_{i=1}^{k} 2(a_{n-1}r_i^{n-1} + a_{n-2}r_i^{n-2} + \dots + a_1 r_i + a_0 - d_i) = 0 \\ \dfrac{\partial \varphi}{\partial a_1} = \sum_{i=1}^{k} 2(a_{n-1}r_i^{n-1} + a_{n-2}r_i^{n-2} + \dots + a_1 r_i + a_0 - d_i) r_i = 0 \\ \dots\dots\dots\dots \\ \dfrac{\partial \varphi}{\partial a_{n-1}} = \sum_{i=1}^{k} 2(a_{n-1}r_i^{n-1} + a_{n-2}r_i^{n-2} + \dots + a_1 r_i + a_0 - d_i) r_i^{n-1} = 0 \end{cases} \tag{6.8}$$

By separating the unknowns and the observations on each side of the equation, the above equation group is rearranged as follows:

$$\begin{cases} a_{n-1}\sum_{i=1}^{k} r_i^{n-1} + a_{n-2}\sum_{i=1}^{k} r_i^{n-2} + \dots + a_1\sum_{i=1}^{k} r_i + a_0\sum_{i=1}^{k} 1 = \sum_{i=1}^{k} d_i \\ a_{n-1}\sum_{i=1}^{k} (r_i^{n-1} r_i) + a_{n-2}\sum_{i=1}^{k} (r_i^{n-2} r_i) + \dots + a_1\sum_{i=1}^{k} (r_i r_i) + a_0\sum_{i=1}^{k} (1 \cdot r_i) = \sum_{i=1}^{k} (d_i r_i) \\ \dots\dots\dots \\ a_{n-1}\sum_{i=1}^{k} (r_i^{n-1} r_i^{n-1}) + a_{n-2}\sum_{i=1}^{k} (r_i^{n-2} r_i^{n-1}) + \dots + a_1\sum_{i=1}^{k} (r_i r_i^{n-1}) + a_0\sum_{i=1}^{k} (1 \cdot r_i^{n-1}) = \sum_{i=1}^{k} (d_i r_i^{n-1}) \end{cases} \tag{6.9}$$

Then, we rewrite the above equation group in the form of matrix as follows:

$$\begin{bmatrix} \sum_{i=1}^{k} (r_i^{n-1} r_i^{n-1}) & \sum_{i=1}^{k} (r_i^{n-2} r s_i^{n-1}) & \dots & \sum_{i=1}^{k} (r_i r_i^{n-1}) & \sum_{i=1}^{k} (1 \cdot r_i^{n-1}) \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{i=1}^{k} (r_i^{n-1} r_i) & \sum_{i=1}^{k} (r_i^{n-2} r s_i) & \dots & \sum_{i=1}^{k} (r_i r_i) & \sum_{i=1}^{k} (1 \cdot r_i) \\ \sum_{i=1}^{k} r_i^{n-1} & \sum_{i=1}^{k} r_i^{n-2} & \dots & \sum_{i=1}^{k} r_i & \sum_{i=1}^{k} 1 \end{bmatrix} \cdot \begin{bmatrix} a_{n-1} \\ \dots \\ \dots \\ \dots \\ \dots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{k} (d_i r_i^{n-1}) \\ \dots \\ \sum_{i=1}^{k} (d_i r_i) \\ \sum_{i=1}^{k} d_i \end{bmatrix} \tag{6.10}$$

Finally, the coefficient vector $\vec{a}$ is obtained by multiplying the inverse of the above left-most squared matrix on both sides of the equation:

$$\vec{a} = \begin{bmatrix} a_{n-1} \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{k}(r_i^{n-1}r_i^{n-1}) & \sum_{i=1}^{k}(r_i^{n-2}r_i^{n-1}) & \cdots & \sum_{i=1}^{k}(r_i r_i^{n-1}) & \sum_{i=1}^{k}(1\cdot r_i^{n-1}) \\ \cdots\cdots & \cdots\cdots & \cdots & \cdots\cdots & \cdots\cdots \\ \sum_{i=1}^{k}(r_i^{n-1}r_i) & \sum_{i=1}^{k}(r_i^{n-2}r_i) & \cdots & \sum_{i=1}^{k}(r_i r_i) & \sum_{i=1}^{k}(1\cdot r_i) \\ \sum_{i=1}^{k} r_i^{n-1} & \sum_{i=1}^{k} r_i^{n-2} & \cdots & \sum_{i=1}^{k} r_i & \sum_{i=1}^{k} 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \sum_{i=1}^{k}(d_i r_i^{n-1}) \\ \cdots\cdots \\ \sum_{i=1}^{k}(d_i r_i) \\ \sum_{i=1}^{k} d_i \end{bmatrix}$$

$$= \left(\begin{bmatrix} r_1^{n-1} & r_1^{n-2} & \cdots & r_1 & 1 \\ r_2^{n-1} & r_2^{n-2} & \cdots & r_2 & 1 \\ \cdots\cdots & & & & \\ r_k^{n-1} & r_k^{n-2} & \cdots & r_k & 1 \end{bmatrix}^T \cdot \begin{bmatrix} r_1^{n-1} & r_1^{n-2} & \cdots & r_1 & 1 \\ r_2^{n-1} & r_2^{n-2} & \cdots & r_2 & 1 \\ \cdots\cdots & & & & \\ r_k^{n-1} & r_k^{n-2} & \cdots & r_k & 1 \end{bmatrix} \right)^{-1} \left(\begin{bmatrix} r_1^{n-1} & r_1^{n-2} & \cdots & r_1 & 1 \\ r_2^{n-1} & r_2^{n-2} & \cdots & r_2 & 1 \\ \cdots\cdots & & & & \\ r_k^{n-1} & r_k^{n-2} & \cdots & r_k & 1 \end{bmatrix}^T \cdot \begin{bmatrix} d_1 \\ d_2 \\ \cdots\cdots \\ d_k \end{bmatrix} \right) \quad (6.11)$$

$$= \left(X^T X\right)^{-1}\left(X^T \vec{y}\right)$$

Proof ends.

6.3 Multiple Quickest Path Computation

An overlay network is constructed with NetLets daemons and virtual links connecting them. Each virtual link is essentially a physical path in the Internet, which usually consists of multiple actual links between underlying routers. We use the active measurement based method to estimate the available path bandwidth and the minimum end-to-end delay for each virtual link. Note that one virtual link always has different measurements in its two-way directional message transmission because the routing in the Internet is asymmetric. However, since we calculate the round-trip time and divided it by two to measure the end-to-end delay, the difference of the two-way directional estimations will not be significant. A typical overlay network with estimated available path bandwidths and minimum end-to-end delays is shown in Figure 6.3 just for the purpose of illustration.

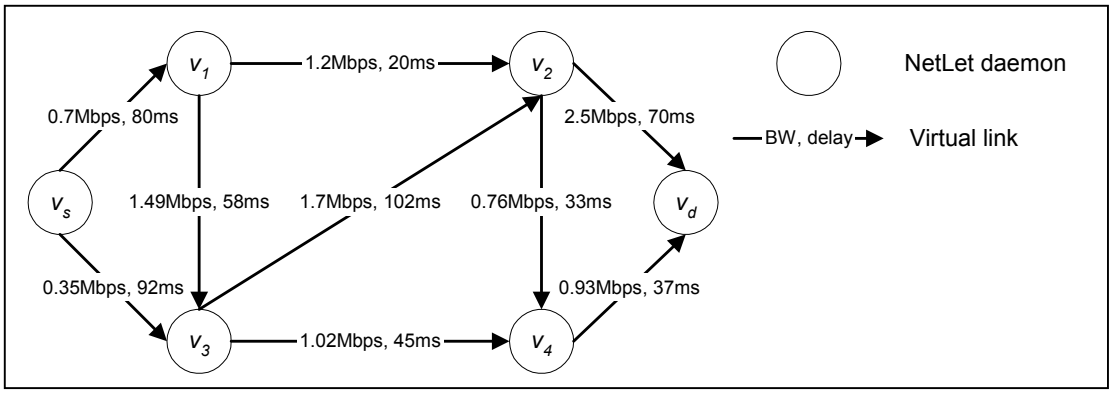


Figure 6.3 Overlay network with estimated path bandwidths and end-to-end delays

An overlay network can be represented by a graph $G(V, E)$ with vertices denoting NetLets daemons and edges denoting virtual links. Each edge in an overlay network graph is not only associated with the minimum delay, which corresponds to the edge

length in a conventional graph, but the bandwidth. The quickest path problem is to find a routing path in an overlay network graph G such that the end-to-end delay time required to send a message of size r from a source node v_s to a destination node v_d is minimum. A routing path in an overlay network is usually made up of one or more virtual links or physical paths in the underlying network. Since the end-to-end delay for message transmission over a routing path does not only depend on the minimum delays of virtual links, but also the associated available bandwidths, apparently the well-known shortest path algorithm, Dijkstra's algorithm, cannot be directly applied to such graphs.

We design an approximate algorithm *MQP*(overlay network graph $G(V, E)$, message size r, path number m, source node v_s, destination node v_d) to compute multiple quickest paths from source node v_s to destination node v_d in an overlay network graph $G(V, E)$. This algorithm is modified from Dijkstra's algorithm [CLR00, RB99] with the new key value of node $v_i, i = s, 1, 2, \ldots, d$ defined by $\frac{r}{ABW[v_i]} + d_{\min}[v_i]$, where $ABW[v_i]$ is the available path bandwidth of the route from source node v_s to node v_i, and $d_{\min}[v_i]$ is the sum of the minimum virtual link delays along the route from source node v_s to node v_i. For simplicity, we use the minimum of the component virtual link bandwidths to approximate $ABW[v_i]$.

The multiple quickest paths algorithm *MQP*(G, r, m, v_s, v_d) is briefly presented in Figure 6.4. Note that every time a quickest path is found, its path bandwidth is subtracted from the bandwidths of its component virtual links before we search for the next quickest path.

MQP(G, r, m, v_s, v_d)
Begin
For each quickest path numbered from 1 to m
  Initialize resolved/unresolved node set, predecessor node list, and key value of source node v_s;
  Compute key value $\frac{r}{ABW[v_i]} + d_{\min}[v_i]$ for node $v_i \neq v_s$;
  While unresolved node set is not empty
    Select the node with the minimum key value;
    Relax its neighbor nodes;
    Remove this node from unresolved node set and add to resolved node set;
  Build the quickest path from v_s to v_d using the predecessor node list;
  Subtract path bandwidth from all component virtual link bandwidths of the current quickest path;
End

Figure 6.4 Algorithm MQP(s, m) for computing multiple quickest paths

The relaxation procedure in algorithm *MQP* is similar to the one used in Dijkstra's algorithm except that *MQP* uses the new key value and the path bandwidth is recomputed for each neighboring node to make sure that its path bandwidth is always the minimum of the component virtual link bandwidths. For implementation details, please refer to the algorithm source code provided in Appendix A.

6.4 Framework of NetLet Daemon

An overview of the NetLet daemon framework is illustrated in Figure 6.5. The NetLet daemon consists of four main function units: end-to-end delay measurement, statistical estimator, multiple quickest path computation, and data routing. The NetLet daemon has one interface with the host process for data delivery and three interfaces with the other active NetLets daemons for end-to-end delay measurement, link information exchange as well as data receiving and forwarding.

The end-to-end delay in an overlay network is collected through active measurements. Specifically speaking, the selected nodes send a set of messages of various sizes to the rest of the nodes in the overlay network and calculate the round-trip time upon receiving each message echoed back by the other nodes. Linear regression is then performed on these end-to-end message transmission delay measurements to obtain the estimates of available virtual link bandwidths and minimum virtual link delays. The virtual link information is exchanged among all participating NetLets daemons to build a complete topology of the overlay network, based on which the routing table is computed using multiple quickest paths algorithm *MQP* at each node.

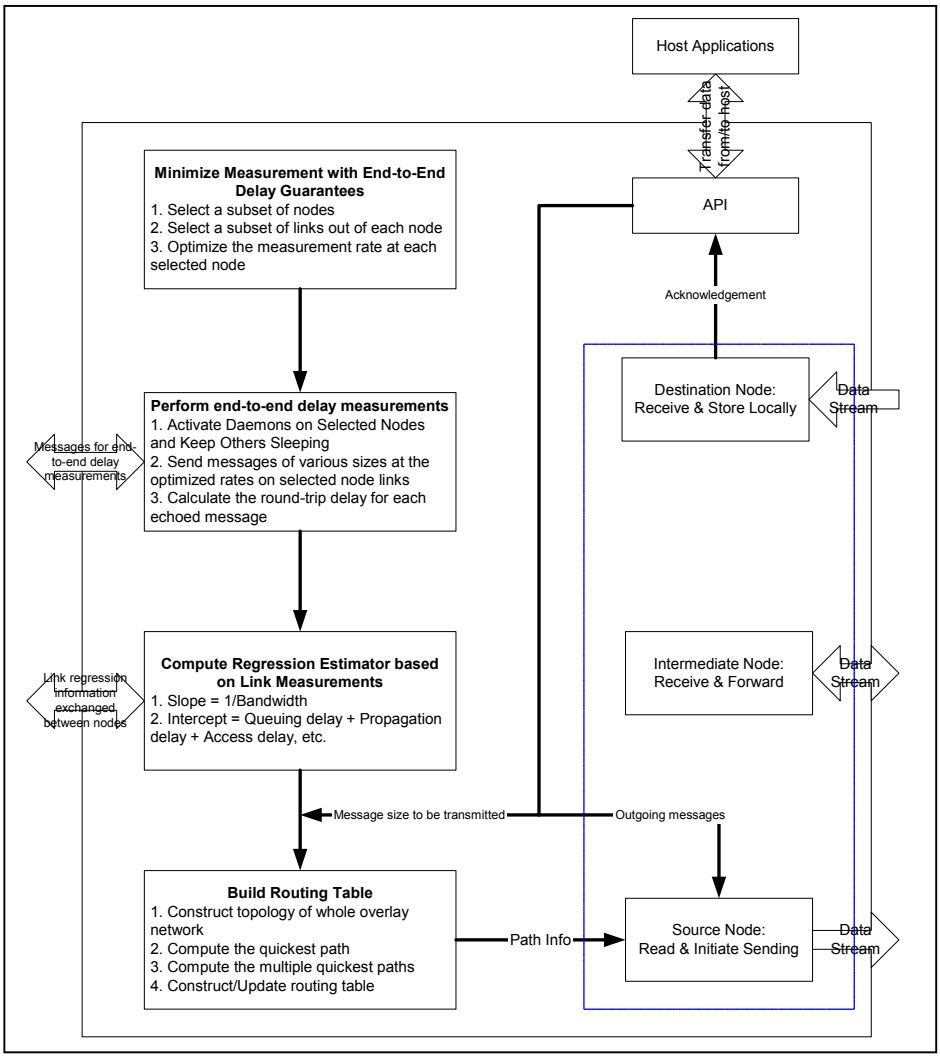


Figure 6.5 Framework of NetLet daemon

As for the data transmission, the source node first retrieves multiple quickest paths from the routing table, divides the data into multiple parts appropriately, and then sends them to the second nodes through different paths concurrently. Suppose that *m* quickest paths have been found: $P_1, P_2, \ldots, P_m$. The partitioning of user data of size *r* into *m* parts: $r_1, r_2, \ldots, r_m$, requires that the transmission times along all routing paths are equal:

$$\begin{cases} \dfrac{r_1}{ABW(P_1)} + d(P_1) = \dfrac{r_2}{ABW(P_2)} + d(P_2) \\ \dfrac{r_2}{ABW(P_2)} + d(P_2) = \dfrac{r_3}{ABW(P_3)} + d(P_3) \\ \cdots\cdots \\ \dfrac{r_{m-1}}{ABW(P_{m-1})} + d(P_{m-1}) = \dfrac{r_m}{ABW(P_m)} + d(P_m) \end{cases} \tag{6.12}$$

and

$$\sum_{i=1}^{m} r_i = r \tag{6.13}$$

Particularly when the number of quickest paths $m = 2$, we use the following equation to partition the data [Rao01]:

$$\begin{cases} r_1 = \dfrac{ABW(P_1) \cdot r}{ABW(P_1) + ABW(P_2)} + \dfrac{ABW(P_1) \cdot ABW(P_2) \cdot [d(P_2) - d(P_1)]}{ABW(P_1) + ABW(P_2)} \\ r_2 = r - r_1 \end{cases} \tag{6.14}$$

As an overlay router, an intermediate NetLet daemon performs routing function. It receives incoming data component, extracts routing path information, and forwards data component to the very next hop on the routing path. The destination node simply receives data components and stores them locally. If all data components have arrived, an acknowledgement is sent from the destination node to the source node to indicate the end of transmission.

6.5 NetLets Implementation in Overlay Network

The current version of NetLets is implemented in C++ on Linux operating system. Both end-to-end measurements and data transmissions use TCP connections. Figure 6.6 shows the detailed NetLet activity diagram.

We have deployed NetLet daemons on the following sites: LSU, ORNL, Purdue, UFL, and GaTech. The topology of the overlay network of NetLet daemons is shown in Figure 6.7. Since there always exists a physical path connecting any two hosts in the Internet via a certain number of intermediate routers, we abstract the real network connection as a complete graph of 5 nodes and 10 edges.

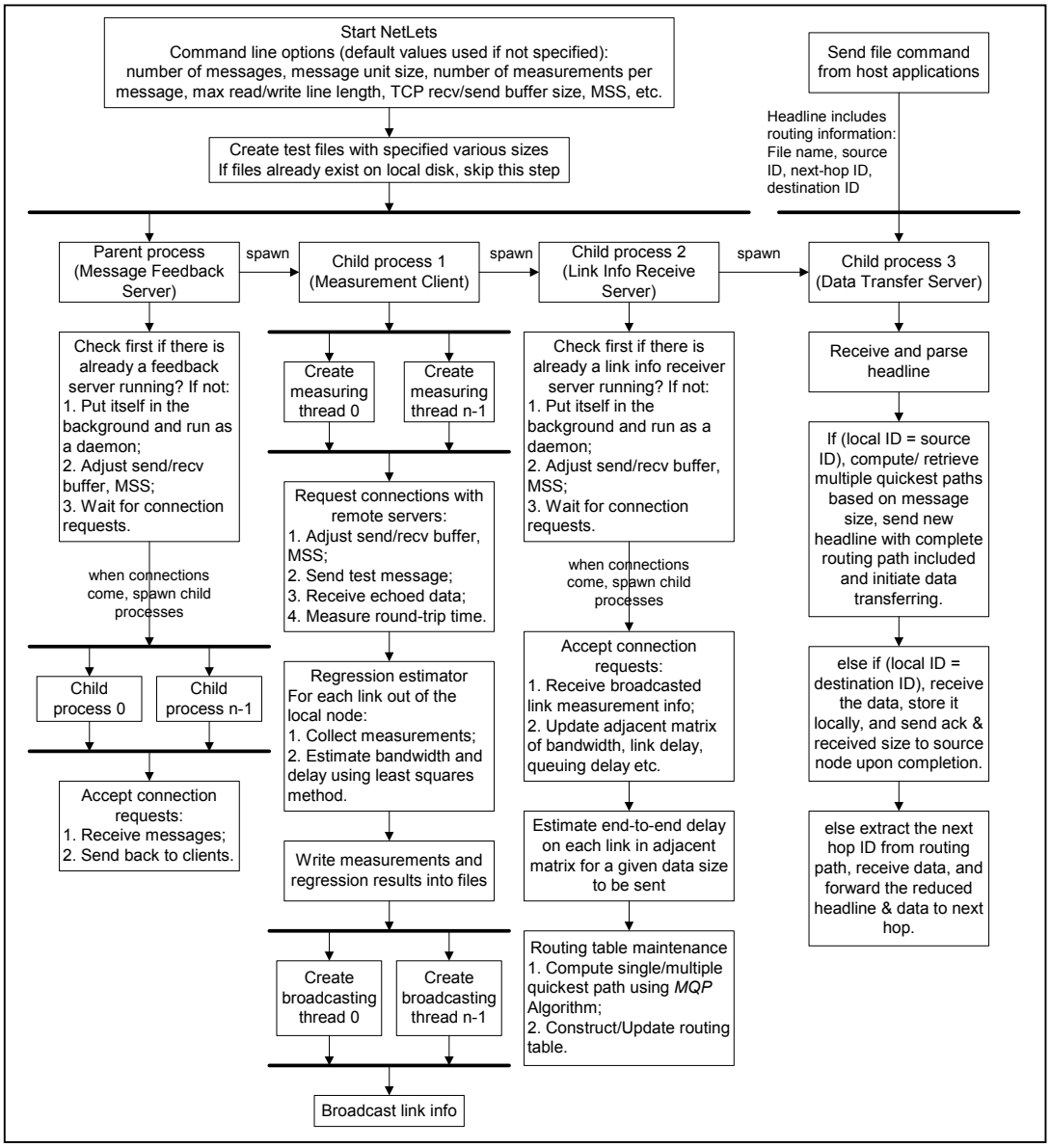


Figure 6.6 NetLets activity diagram

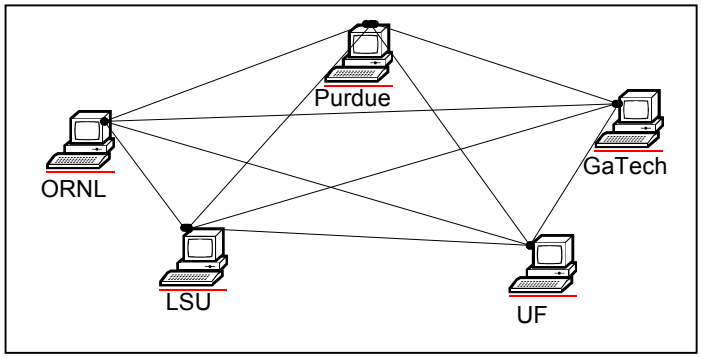


Figure 6.7 Overlay network topology of NetLet daemons deployed over Internet

We design the NetLets daemons in such a way that all measurement control parameters are tunable, such as TCP socket buffer size, MSS, test message sizes, number of measurements per size, the unit of data written and read, etc. to achieve better network performances. Some of the socket options may significantly affect the network performance depending upon the transmission distance, message size, and effective bandwidth, etc. The performance comparison between the NetLet with the default socket options and the one with the adjusted socket options is shown in Figure 6.8, which demonstrates that a properly tuned socket send/receive buffer has reduced the end-to-end message transmission delay to nearly two thirds of the previous bandwidth measurement without buffer tuning. All the delay measurements used in the comparison are collected from the NetLets deployed at LSU and ORNL.

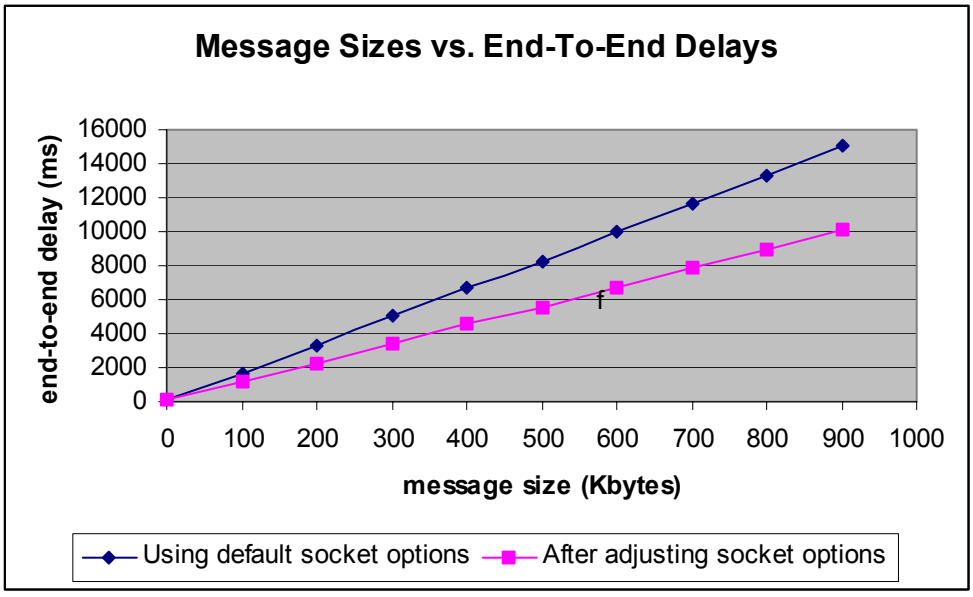


Figure 6.8 Impact of socket buffer tuning on network performance

CHAPTER 7 CTT PROTOCOL FOR WIRELESS NETWORKS

7.1 Introduction

In the previous chapters, we have discussed the new transport control protocol TCOU using stochastic approximation methods as well as overlay networks of NetLets. In this chapter, we consider the transmission control problem in dynamic wireless networks and adapt ONTCOU to wireless environments with regard to the wireless-specific connectivity issue.

Wireless networks have been deployed in a wide spectrum of applications ranging from campus access to robot teams to sensor networks [Perkins01]. Especially, the study of detection, surveillance, and tracking systems has been an active area of research since early 90s. Recent advances in the sensor technology make it possible to use multiple sensors of different types in both military and civilian applications. The wireless network is usually the only feasible way of communication among moving computing devices, sensors or robots, especially when the environment is harsh, unreliable, or even adversarial. Depending upon the implementation environments and application purposes, wireless networks operate under either access point or ad-hoc modes.

Access point mode is typically used in some structured areas where installing network infrastructures is possible and the moving distance of network nodes remains in a limited region, such as campus and corporation buildings. In this case, the access points are able to cover all node movements so that the node connections through access points essentially form a conventional LAN [Stallings01]. However, the communication in a wireless network operated under ad-hoc mode cannot be simply handled in the same way by the conventional communication scheme. An ad-hoc wireless network does not provide any specially designed routing hardware and software, and its link connectivity through wireless radio is highly dynamic and unpredictable due to the unstructured nature of the terrain and the distant movements of network nodes. A robot team equipped only with IEEE 802.11 wireless cards and widely dispersed in an unstructured environment serves as a good example of such networks [RWIM02]. The communication among moving robots via conventional TCP byte-streams may experience difficulties that are not involved in wired networks.

As it is well known, the data transfer using TCP channel needs the support of underlying routers and requires a direct or indirect connection between the source and destination nodes existing during the whole period of transmission. Obviously, these requirements may not be sufficiently satisfied in the ad-hoc mode wireless network due to its operational characteristics. Furthermore, the usual implementation of TCP congestion control strategies is not suited for the wireless environment because TCP always interprets a packet loss as a network congestion signal and reduces its sending rate. As a matter of fact, in wireless networks the packet loss is mostly due to physical link failures so that the opposite packet loss handling mechanism is actually needed. In other words, the source rate must be increased to account for the packet loss.

To meet the above challenges, we shall adapt ONTCOU to wireless environments by developing and implementing an appropriate transmission control mechanism and a class of protocols based on Connectivity-Through-Time algorithm.

7.2 Connectivity-Through-Time Concept

The connectivity of an ad-hoc wireless network can be represented by a graph $G(V, E(t))$, where V represents the set of network nodes and $E(t)$ represents the set of direct wireless communication links between any two nodes at time t. A network graph of five mobile nodes is illustrated in Figure 7.1, where a two-direction arrow represents a direct wireless link between two nodes whose physical distance is less than the maximum wireless radiation distance.

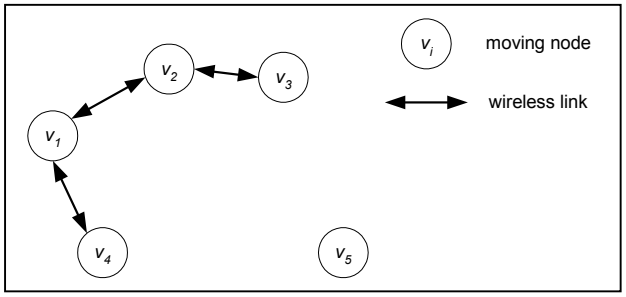


Figure 7.1 Ad-hoc wireless network graph

The ad-hoc wireless network is different from the overlay network deployed in the Internet we discussed before in the aspect of connectivity. Any two nodes in the Internet are essentially connected through multiple physical links between underlying routers therefore the communication between two Internet nodes is always available through TCP channels. However, in the ad-hoc wireless network, only two nodes with a direct wireless link can communicate with each other using the default communication stack. Although there exists an indirect connection between node v_3 and node v_4 in Figure 7.1, they are not able to communicate with each other via a default TCP stream because TCP is an end-to-end protocol that does not provide routing and forwarding functions. Furthermore, node v_5 is currently isolated because it is out of the maximum wireless radiation distance from any other nodes.

Since the NetLets, the building block of ONTCOU, are designed to work as overlay routers, the communication between two nodes with an indirect connection in the ad-hoc wireless network can be handled conveniently using the existing overlay network. By introducing a Connectivity-Through-Time (CTT) concept and incorporating its corresponding transmission control protocol into the existing ONTCOU, we are also able to carry out communications between nodes that even have never been connected to each other either directly or indirectly at any time. Figure 7.2 illustrates such a typical example of the Connectivity-Through-Time concept that node v_1 transmits data to node v_5 by the movements of node v_4.

As shown in Figure 7.2, at time t_1 the source node v_1 searches in its neighborhood for the destination node v_5. Since node v_5 is completely unreachable at the moment, node v_1 decides to broadcast data together with destination information to all its neighbor nodes first, one of which, node v_4 receives and stores the data and destination information. Node v_4 then moves towards node v_5 at time t_2 and eventually it enters the area covered by the maximum wireless radiation distance of node v_5 at time t_3. Once a new link is

detected, node v_4 retrieves stored data from its data repository and checks for destination availability. The data is finally transmitted to node v_5 since the destination is now on the neighbor node list of node v_4.

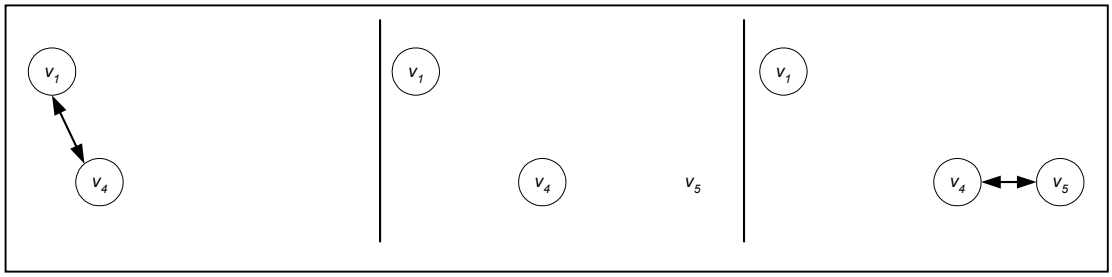


Figure 7.2 Example of Connectivity-Through-Time

7.3 Implementation of CTT Protocol in ONTCOU

The Connectivity-Through-Time protocol has been implemented and integrated into ONTCOU in C++ on Linux. A brief description of the implementation framework is illustrated in Figure 7.3. The connectivity computation module is a relatively independent function unit, which is responsible for maintaining the neighbor list and constructing the routing table. The up-to-date path information is then retrieved from the routing table and provided to the transport control module. The network connectivity is updated upon receipt of a special datagram called IAmHere, which is exchanged among nodes with the neighbor list enclosed. This connectivity computation module is actually a simplified version of virtual link bandwidth measurement in ONTCOU.

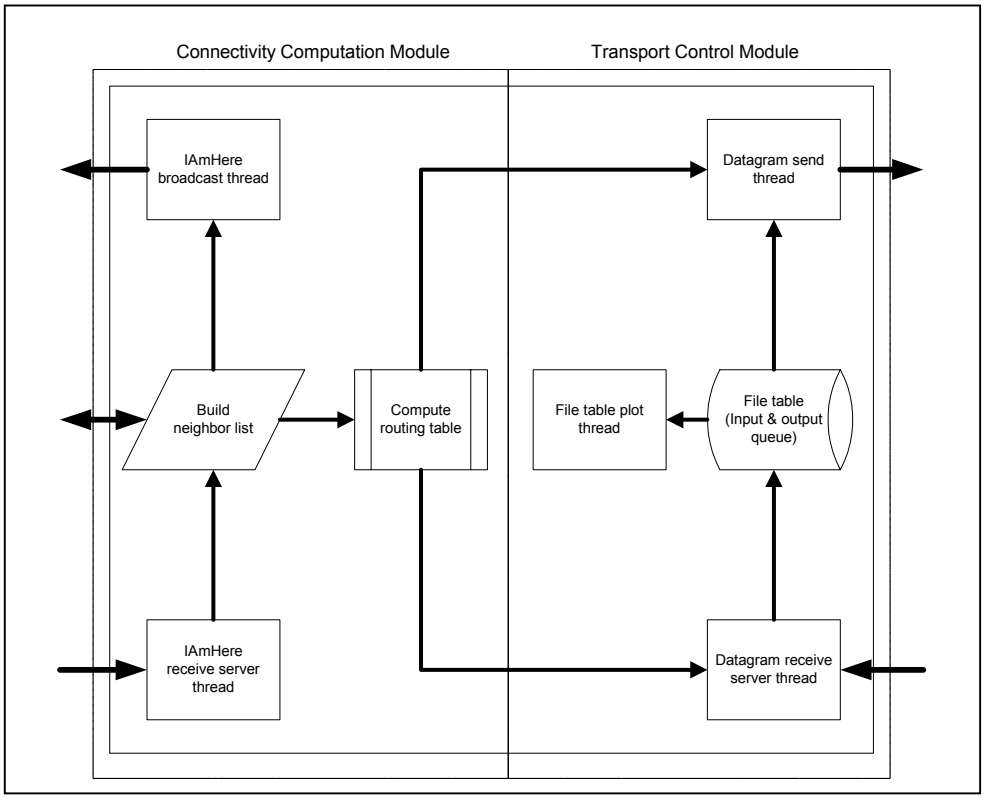


Figure 7.3 Framework of CTT implementation

The datagram receiving unit accepts UDP datagrams either from adjacent nodes or local host. If the arrived message is interpreted as a send command issued by the local host, the designated data source will be read directly from hard disk and packed in fixed-size units with a header attached. Then an appropriate amount of memory space is allocated for a file buffer to hold the newly created datagrams. If the received datagrams are not originated locally, they will be simply placed in the datagram table of the corresponding file buffer.

The datagram sending unit repeatedly scans the whole file list on a sequential basis and determines sending priority for each datagram. Any in-order datagrams destined to the local host are delivered to the host application immediately, otherwise they are held until the holes are filled. When a datagram is selected, it is loaded into outgoing queue for forwarding or broadcasting.

The flow chart of the implementation details is given in Figure 7.4. A datagram is assigned one of the five modes based on the current network condition and its own status: READY, STANDBY, CTT, SENT, ARRIVED.

For a newly created datagram, it is set as READY mode if a direct or indirect path is found between its source and destination; otherwise it goes to STANDBY mode. A passing by datagram remains in READY mode if the local host is on the path and the next hop is reachable. It switches to CTT mode if the next hop is unreachable due to dynamic changes of network connectivity. If a datagram is received by a node that is not expecting it, the datagram enters STANDBY mode. Datagrams in READY mode or CTT mode when the next hop becomes reachable have the highest priority to be selected and put in the outgoing queue, and they change to SENT mode right after they are successfully dispatched. A broadcast as well as path re-computation is enforced when the timeout arrives for a datagram in CTT, STANDY, or SENT mode. When a datagram arrives at its destination, it is set as ARRIVED mode and a special acknowledgment DGARRIVEDACK is broadcasted backwards. Once a DGARRIVEDACK is received at any nodes, the corresponding datagram is removed from the datagram table to release its allocated memory space and a special label is then assigned to indicate that this datagram has been received by the destination.

When the last datagram (not necessarily the one with the last sequence number) arrives at the destination, another type of acknowledgment FILESAVEDACK is broadcasted over the network. The whole datagram table is cleaned up immediately when such an acknowledgment is received no matter the table is complete or incomplete.

An acknowledgment DGPASSBYACK is broadcasted when a datagram reaches a node to which the datagram is not destined. The DGPASSBYACK carries the list of nodes that have received this datagram, which can be used to effectively reduce unnecessary flooding.

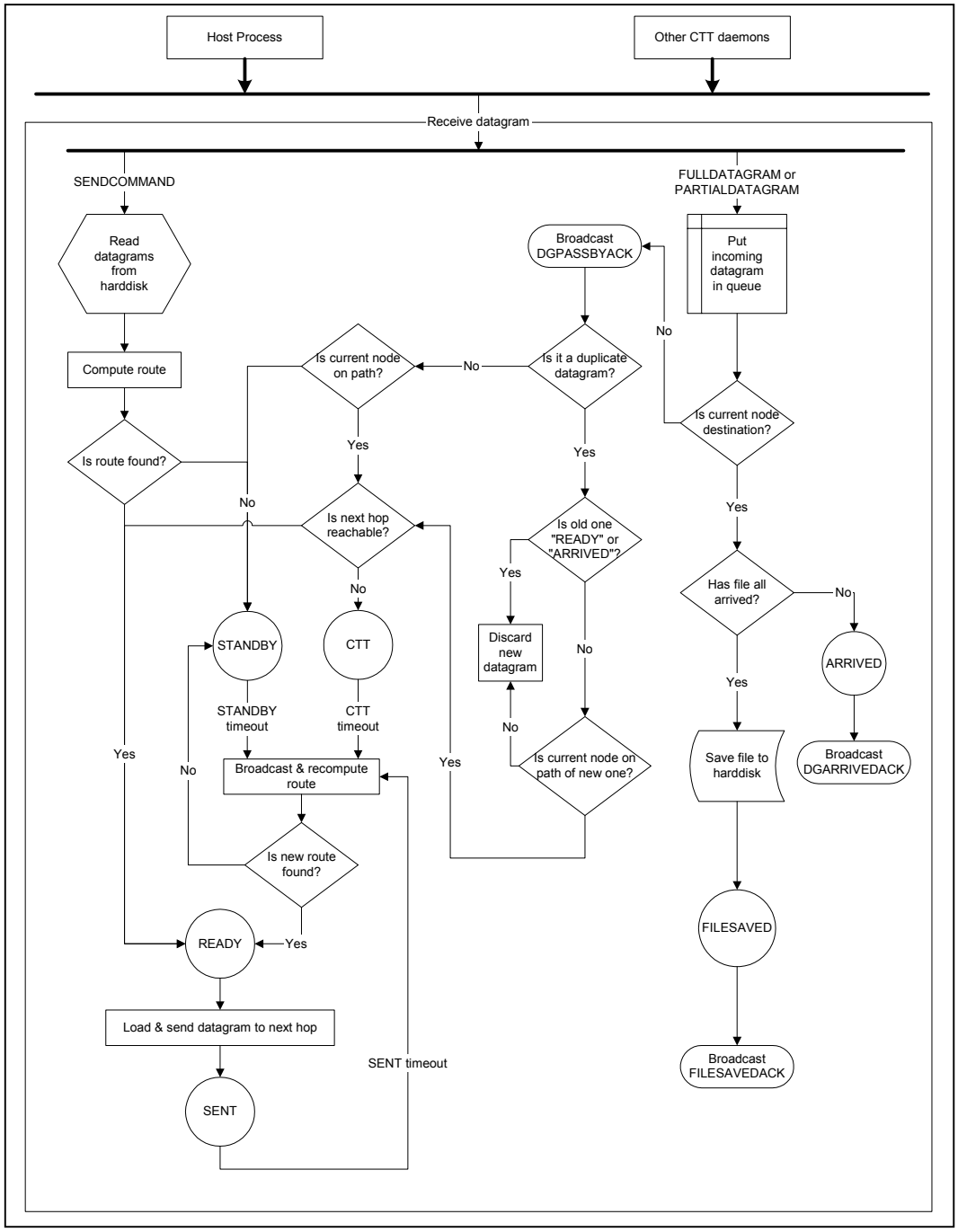


Figure 7.4 Control flow chat of Connectivity-Through-Time protocol

The data structures used in the implementation of Connectivity-Through-Time protocol are shown in Figure 7.5.

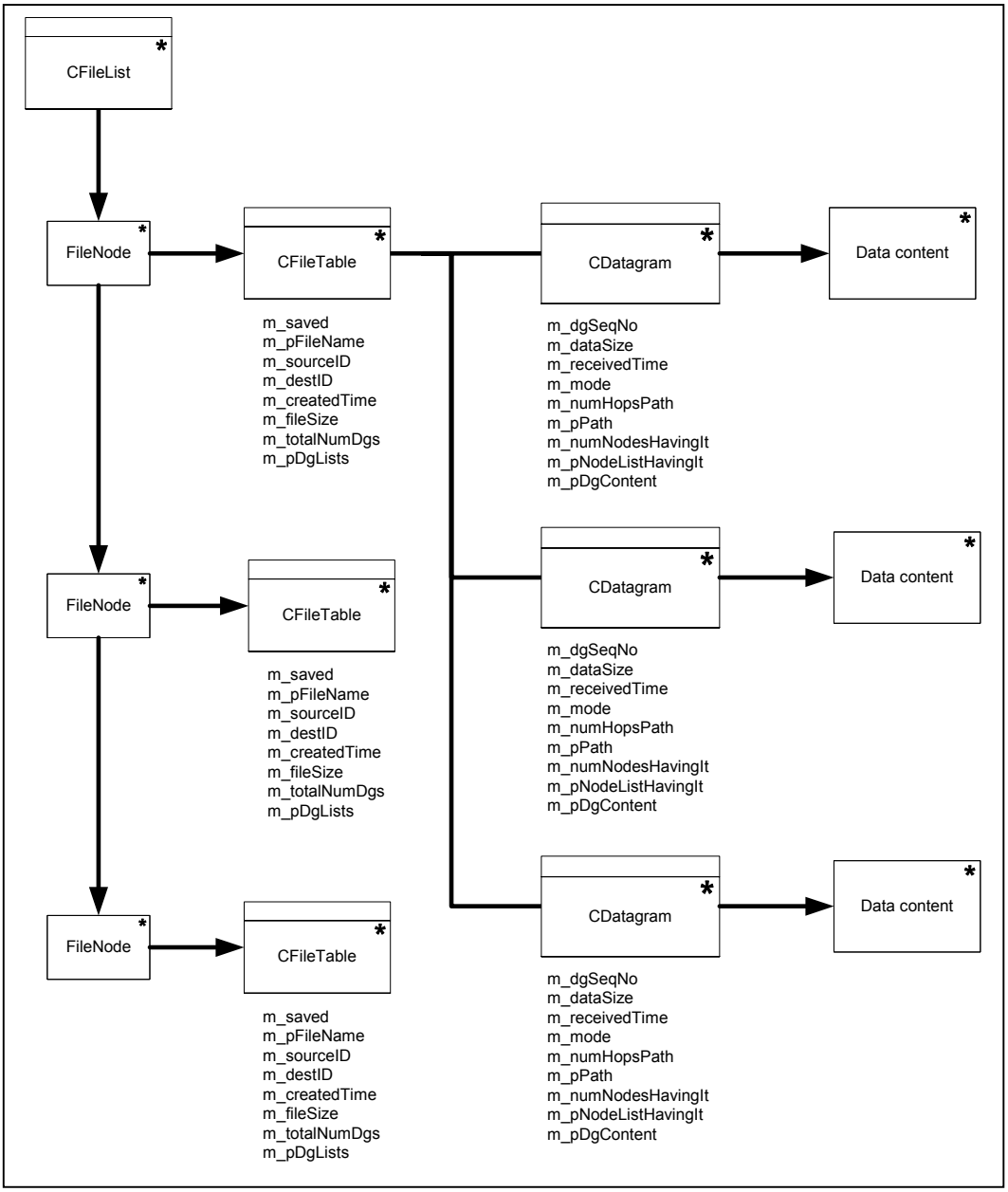


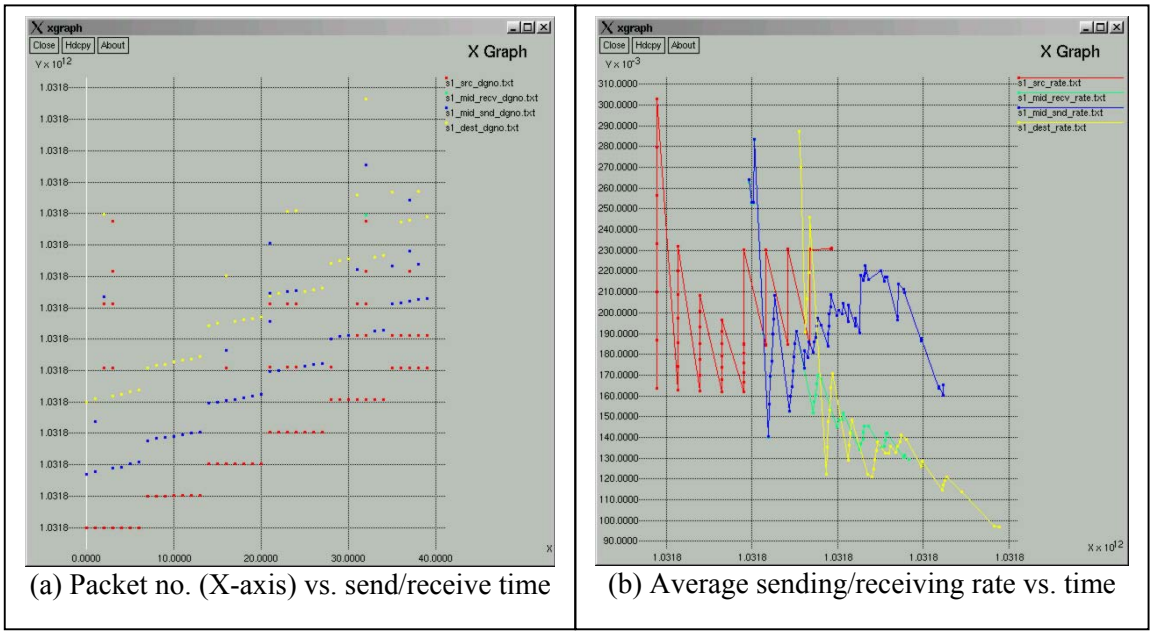
Figure 7.5 Data structures used in the implementation of CTT protocol

7.4 Experimental Results

We describe experimental results based on the implementation on a team of mobile robots. The testing is carried out on a team of four Mini ATRV mobile robots equipped with 802.11 wireless cards.

In scenario one, we demonstrate that the robots are used as routers to deliver messages when there is no direct path from source to destination nodes, but the intermediate node falls within the intersection of the radio ranges of source and destination. Plot (a) of Figure 7.6 shows the datagram sending/receiving time and Plot (b) shows the corresponding datagram sending/receiving rate measured at the source node, the

intermediate node, and the destination node. According to Plot (a) of Figure 7.6, we observed that the intermediate node receives datagrams from the source node and forwards them to the destination node. Since the connections of two hops exist all the time, the datagrams remain in READY mode through the path until they arrive at the destination. Since the incoming and outgoing connections at the intermediate node contend with each other for the physical channel bandwidth, the intermediate and destination nodes have lower receiving rate than the source and intermediate sending rate.

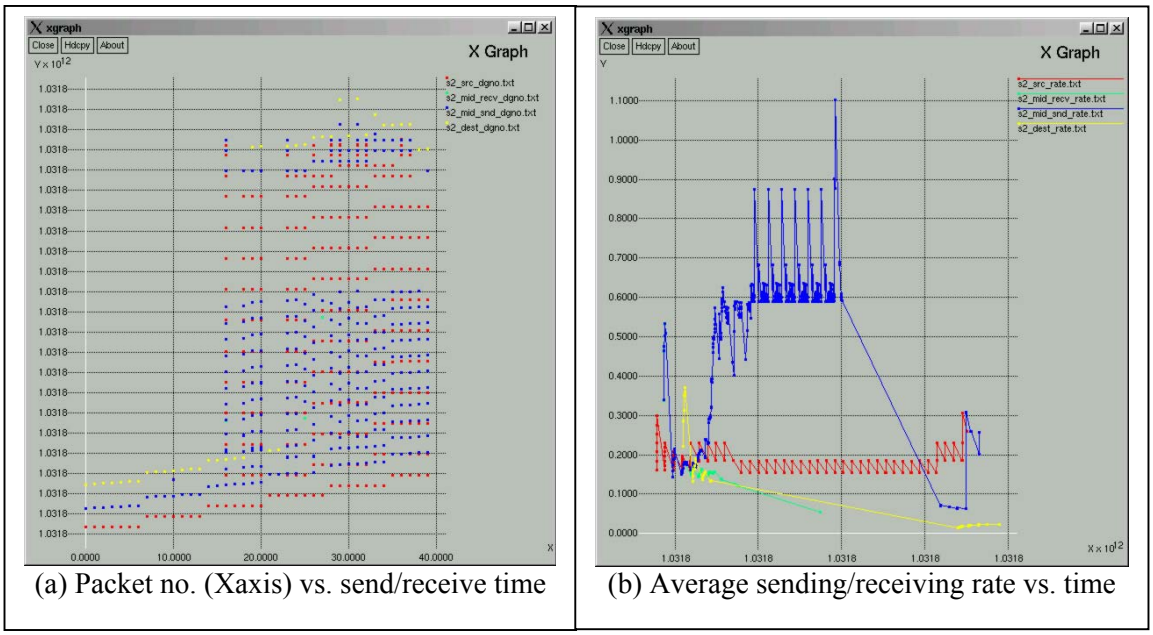


(a) Packet no. (X-axis) vs. send/receive time

(b) Average sending/receiving rate vs. time

Figure 7.6 Scenario 1: Robot serves as a router

In scenario two, we illustrate that messages will be buffered where the path to the destination breaks. Plot (a) of Figure 7.7 shows the datagram sending/receiving time and Plot (b) shows the corresponding datagram sending/receiving rate measured at the source node, the intermediate node, and the destination node. As illustrated in Figure 7.7, the data transmission has three stages. The first stage is similar to scenario one: the source and destination are connected to the intermediate node but there is no direct connection between them. The intermediate node serves as a router receiving and forwarding the first set of datagrams. In the second stage, the connection between the intermediate node and destination breaks, so the intermediate node starts buffering incoming datagrams, which are now in CTT mode waiting for the connection to be brought back up. The second set of datagrams is delivered in the third stage where the connection of the second hop resumes. During the second stage, the throughput of the destination decreases because it does not receive any data from the intermediate node, while the intermediate node has a higher sending rate because the CTT timeout incurs broadcasting.

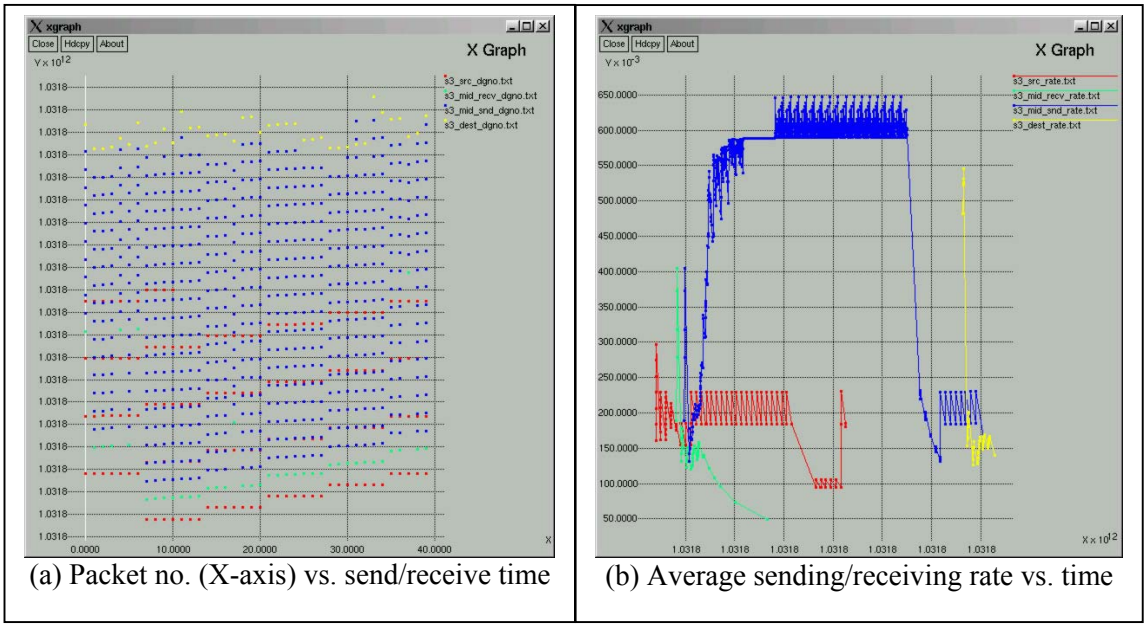


(a) Packet no. (Xaxis) vs. send/receive time

(b) Average sending/receiving rate vs. time

Figure 7.7 Scenario 2: Path to the destination breaks

In scenario three, we show that the messages are delivered between source and destination, which are never connected to each other even via multiple hops. Plot (a) of Figure 7.8 shows the datagram sending/receiving time and Plot (b) shows the corresponding datagram sending/receiving rate measured at the source node, the intermediate node, and the destination node.



(a) Packet no. (X-axis) vs. send/receive time

(b) Average sending/receiving rate vs. time

Figure 7.8 Scenario 3: Messages are delivered through Connectivity-Through-Time

As illustrated in Figure 7.8, this scenario can also be divided into three stages. In the first stage, only the connection between the source and intermediate node exists, and the datagrams are broadcasted after the STANDBY timer expires. This connection breaks in

the second stage where the intermediate node is the only active node performing broadcasts. In the third stage, the connection between the intermediate node and destination comes up so that a new path is found to deliver the datagrams from intermediate node to the destination. As a matter of fact, some of the datagrams are received by the destination through broadcast right after the second hop connection is created and before the new path is computed. Observe from the throughput curves that the destination has almost the same throughput as the intermediate node. The explanation for this observation is that the two hop connections never exist at the same time so that each of them has the exclusive bandwidth utilization at different times.

CHAPTER 8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

Our research efforts have been made to conduct a rigorous analytical study on the performance and design of transport layer protocols. We discussed the problems associated with the default transport layer protocols, TCP and UDP, and systematically applied stochastic approximation methods to the design of new protocols for goodput stabilization and maximization. In this section we make conclusions of our research work presented in this dissertation.

We designed a novel transport contol model and collected network performance measurements over a time span as long as more than half a year. The extensive observation data shows that there are various types of randomness involved in the goodput and delay measurements and the network traffic is stochastic in nature. We also determined that both control parameters in the transport control model have significant effects on the network performance and there are interaction effects between them.

We developed and implemented TCOU, a new class of end-to-end transport control mechanisms based on stochastic approximation methods to solve the goodput stabilization and maximization problems. My personal understanding of leading stochastic approximation methods to the design of transport protocols is to "let nature take its course" and avoid using flow and congestion control parameters, whose values are usually preselected by protocol implementers. These new solutions were mathematically proved to converge nicely under relatively loose conditions, which was further justified by extensive experimental results. These stochastic approximation method-based protocols are expected to improve or even replace the current TCP design methodology.

As for TCOU for goodput stabilization, the control process converges very reliably and smoothly to the desired goodput level, which is reasonably chosen within the first monotonically increasing phase in the goodput response regression. The selection of the starting point affects the convergence speed. Generally speaking, the closer the starting point is selected to the target level, the more quickly the achieved goodput converges to the target level. Therefore, an appropriate starting point must be located around the target level. The gain coefficients also have some impact on the source control process. In general, a bigger step adjustment size produced by the coefficients may cause more intensive oscillation in the control process. Furthermore, our experimental results show that TCOU for goodput stabilization is robust against the presence of various conformant TCP traffic such as HTTP, FTP, and SSH.

As for TCOU for goodput maximization, we have achieved the maximum goodput consistently three times more than the one achieved by default TCP. Same as TCOU for goodput stabilization, the selection of starting point and coefficient values affect the convergence speed as well as the smoothness of the optimization process. The experimental results also show that the fairness problem is implicitly handled by the dynamic version of the selected stochastic approximation method. If more TCP sessions join in the background and the network gets busier, the maximum or transition point in the goodput response regression moves towards left; otherwise, it moves towards right. However, our protocol is able to keep on-line track of the changing trend in the goodput

response regression and adjust the source rate to maintain the goodput around the maximum point.

TCOU has been implemented through overlay networks in the Internet. The overlay network together with the built-in TCOU forms ONTCOU to overcome limitations of default TCP and UDP in the aspects of throughput, fairness, stability, and dynamics. We developed and applied new techniques in the implementation of TCOU, such as floating window based flow control, RTT based packet loss detection, new congestion recovery, etc. Besides, we discussed and solved some conceptual and optimization problems in the overlay network such as bandwidth and delay measurements, multiple quickest paths computation, optimal data routing. The construction of overlay networks makes it possible to set up a bridge between theoretical research and Internet deployment of new transport control protocols.

Since the packet loss in wireless networks does not necessarily indicate network congestion, TCP's AIMD algorithm is not suited for the wireless environment. Furthermore, TCP may not even work in an ad-hoc wireless network with dynamically changing network topology because TCP needs the support from underlying routing facilities and requires a continous connection during the whole period of transmission. We developed and integrated the Connectivity-Through-Time protocol into ONTCOU to resolve these wireless-specific transmission and connectivity issues. The CTT protocol was implemented and tested on a team of mobile robots and the experimental results from various scenarios have illustrated the effectiveness of the protocol.

8.2 Future Work

The research presented in this dissertation is focused on the design of new transport control protocols based on stochastic approximation methods, which open a new and promising research direction for network transport control. The salient features of the new protocols provide great potential for Internet applications, and deserve further research efforts. We may further explore theoretical and implementation aspects:

The validity of the assumptions for the convergence of TCOU based on dynamic RMSA and SPSA methods needs to be assessed and considered in greater depth within the context of diverse network conditions. As the experimental results show, the selection of different starting points affects the convergence speed and oscillation intensity of the converging process in TCOU for both goodput stabilization and goodput maximization. We expect to develop a systematic way of choosing an appropriate starting point based on the offline network performance measurements. In the current design of TCOU for goodput maximization, the goodput response is the only output measurement we use in the recursive update procedure for the source rate control. There is a possibility that TCOU could become trapped in the second unstable phase of the goodput response curve if there is a rapid and massive increase in the background traffic. In other words, TCOU may diverge if the adjustment pace of the control step is slower than the variation of the dynamic regression profile. One possible way to make TCOU work more stable is to consider the loss rate response concurrently in the rate control process to more adaptively adjust the control parameters.

Many techniques used in the implementation of TCOU can be further improved. For instance, currently the acknowledgement is based on every datagram that is successfully received by the destination. Instead, we may explore the advantage of using a delayed acknowledgement scheme. In the current implementation of overlay network, the NetLets

daemons still use default TCP to estimate the available path bandwidth. We expect that eventually TCOU can be substituted for all TCP-based functions in ONTCOU.

BIBLIOGRAPHY

[ABKM01] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, R. Morris. Resilient Overlay Networks. *Proc. 18th ACM SOSP*, Banff, Canada, October 2001.

[Andersen01] D.G. Andersen. Resilient Overlay Networks. MS Thesis, Massachusetts Institute of Technology, May 2001.

[APS99] M. Allman, V. Paxson, W. Stevens. TCP Congestion Control. IETF RFC 2581, April 1999.

[Balakrishnan98] H. Balakrishnan. Challenges to Reliable Data Transport over Heterogeneous Wireless Networks. Ph.D. thesis, Computer Science Division, Univ. of California at Berkeley, Berkeley, CA, August 1998.

[BMP87] A. Benveniste, M. Metivier, P. Priouret. Adaptive Algorithms and Stochastic Approximations. Masson, Paris, 1987, Springer-Verlag Berlin Heidelberg, 1990.

[BP95] L.S. Brakmo, L.L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[Braden89] R. Braden. Requirements for Internet Hosts – Communication Layers. IETF RFC 1122, October 1989.

[Chung54] K.L. Chung. On a Stochastic Approximation Method. *Annals of Mathematical Statistics*, 25, pp. 463-483, 1954.

[CJ89] D.M. Chiu, R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 1989.

[CK74] V. Cerf, R. Kahn. A Protocol for Packet Network Interconnection. *IEEE Transactions on Communications*, vol. COM-22, pp.637-648, May 1974.

[Clark88] D.D. Clark. The Design Philosophy of the DARPA Internet Protocols. *Proceedings of SIGCOMM '88 Conference*, ACM, pp. 106-114, 1988.

[CLR00] T.H. Cormen, C.E. Leiserson, R.L. Rivest. Introduction to Algorithms. the MIT Press, 2000.

[CM01] J. Curtis, T. McGregor. Review of Bandwidth Estimation Techniques. *New Zealand Computer Science Research Students' Conference*, 19-20th April 2001, University of Canterbury, New Zealand.

[Comer98] D.E. Comer. Computer Networks and Internets, Prentice Hall PTR, 1998.

[CPW98] S. Cen, C. Pu, J. Walpole. Flow and Congestion Control for Internet Streaming Applications. *Proceedings of Multimedia Computing and Networking 1998*, January 1998.

[DRM02] C. Dovrolis, P. Ramanathan, D. Moore. Packet Dispersion Techniques and Capacity Estimation. Submitted to the IEEE/ACM Transactions in Networking, 2002.

[Dupac65] V. Dupac. A Dynamic Stochastic Approximation Method. *Annals of Mathematical Statistics*, 36(6): 1695-1702, 1965.

[FB98] J.D. Faires, R. Burden. Numerical Methods, second edition. Brooks/Cole Publishing Company, 1998.

[FF99] S. Floyd, K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 1999.

[FJ92] S. Floyd, V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3): 115-156, Sep. 1992.

[FJ93] S. Floyd, V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4): 397-413, August 1993. ftp://ftp.ee.lbl.gov/papers/early.ps.gz.

[Floyd91] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. *ACM computer Communication Review*, 21(5): 30-47, Oct. 1991. URL: http://www-nrg.ee.lbl.gov/nrg-papers.html/.

[FW97] R.J. Freund, W.J. Wilson. Statistical Methods, revised edition. Academic Press, 1997.

[Haykin00] S. Haykin. Communication Systems, 4th edition. John Wiley & Sons, Inc., 2000.

[Hypercast] http://www.cs.virginia.edu/~mngroup/hypercast/general.html.

[Jacobson88] V. Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM '88*, pages 314-329, August 1988.

[JD02] M. Jain, C. Dovrolis. End-to-End Available Bandwidth: Measurement methodology, Dynamics, and Relation with TCP Throughput. *Proceedings of ACM SIGCOMM*, August 2002.

[JE96] S. Jacobs, A. Eleftheriadis. Providing Video Services Over Networks without Quality of Service Guarantees. *Proceedings of World Wide Web Consortium Workshop on Rea-time Multimedia and the Web*, October 1996.

[JGJKO00] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. *Proceedings of 4th USENIX OSDI*, October 2000.

[Johnson95] S.R. Johnson. Increasing TCP Throughput by Using an Extended Acknowledgement Interval. MS thesis, the College of Arts and Sciences of Ohio University, 1995.

[KC78] H.J. Kushner, D.S. Clark. Stochastic Approximation Methods for Constrained and Unconstrained Systems. Springer-Verlag New York Inc., 1978.

[Kelly99] F. Kelly. Mathematical Modeling of the Internet. *Proceeding of International Congress on Industrial and Applied Mathematics*, 1999. http://www.statslab.cam.ac.uk/~frank/mmi.html.

[KMT98] F.P. Kelly, A. Maulloo, D. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of Operations Research Society*, 49(3): 237-252, March 1998.

[KS00] S. Kunniyur, R. Srikant. End-to-end Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks. *Proceedings of IEEE INFOCOM*, March 2000.

[KW52] J. Kiefer, J. Wolfowitz. Stochastic Estimation of the Maximum of a Regression Function. *Annals of Mathematical Statistics*, 23, 1952, 462-466.

[KY97] H.J. Kushner, G.G. Yin. Stochastic Approximation Algorithm and Applications. Springer-Verlag, New York, Inc. 1997.

[LL99] S.H. Low, D.E. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. *IEEE/ACM Transactions on Networking*, 7(6): 861-874, December 1999.

[LPD02] S.H. Low, F. Paganini, J.C. Doyle. Internet Congestion Control. *IEEE Control Systems Magazine*, February 2002.

[LPW02] S.H. Low, L.L. Peterson, L. Wang. Understanding Vegas: A Duality Model. *Journal of the ACM*, Vol. 49, No. 2, March 2002, Pages 207-235.

[MF97] J. Mahdavi, S. Floyd. TCP-friendly Unicast Rate based Flow Control. Note sent to the end2end-interest mailing list, 1997.

[MLAW99] J. Mo, R. La, V. Anantharam, J. Walrand. Analysis and Comparison of TCP Reno and Vegas. *Proceedings of IEEE INFOCOM*, March 1999.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. TCP Selective Acknowledgement Options. IETF RFC 2018, October, 1996.

[Net100] http://www.net100.org.

[Paxson96] V. Paxson. End-to-End Routing Behavior in the Internet. *Computer Communication Review*, vol. 26, no. 4, pp. 25-38, Oct. 1996.

[PD99] L.L. Peterson, B.S. Davie. Computer Networks: A Systems Approach. Morgan Kaufmann Publishers, 2nd edition, October 1999.

[Perkins01] C. Perkins, editor. Ad Hoc Networking, Addison-Welsey, 2001.

[PKTK99] J. Padhye, J. Kurose, D. Towsley, R. Koodli. A Model based TCP-friendly Rate Control Protocol. *Proceedings of Network and Operating Systems Support for Digital Audio and Video 1999*, June 1999.

[Postel81] J. Postel. Transmission Control Protocol. IETF RFC 793, September 1981.

[Rao01] N.S.V. Rao. NetLets: End-to-end QoS Mechanisms for Distributed Computing Over Internet Using Two-paths. *International Conf. on Internet Computing*, 2001.

[RB99] N.S.V. Rao, S.G. Batsell. Algorithm for Minimum End-to-End Delay Paths. *IEEE Communications Letters*, 1999.

[RC02] N.S.V. Rao, L.O. Chua. On Dynamics of Network Transport Protocols. *Proceedings of Workshop on Signal Processing, Communications, Chaos and Systems*, 2002.

[RGBR00] N.S.V. Rao, W.G. Grimmell, Y. Bang, S. Radhakrishnan. Quickest Paths for Different Network Router Mechanisms. ORNL Technical Report, TM-2000/208, 2000.

[RM51] H. Robbins, S. Monro. A Stochastic Approximation Method. *Annals of Mathematical Statistics*. 22, 1951, 400-407.

[RRCWI01] N.S.V. Rao, S. Radhakrishnan, B.Y. Cheol, Q. Wu, S.S. Iyengar. NetLets for Measurement-based Routing for End-to-End Performance Over the Internet. *Computer Communications*, 2001.

[RWIM02] N.S.V. Rao, Q. Wu, S.S. Iyengar, A. Manickam. Connectivity-Through-Time Protocols for Dynamic Wireless Networks to Support Mobile Robot Teams. *2003 IEEE International Conference on Robotics and Automation*, Taiwan, 2003.

[Sabul] http://www.dataspaceweb.net/sabul.htm.

[SAS] http://www.sas.com.

[Spall92] J.C. Spall. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Transactions on Automatic Control*, 37: 332-341, 1992.

[Spall94] J.C. Spall. Developments in Stochastic Optimization Algorithms with Gradient Approximations Based on Function Measurements. *Proceedings of the 1994 Winter Simulation Conference*, 1994.

[Spall97] J.C. Spall. A One-measurement Form of Simultaneous Perturbation Stochastic Approximation. *Technical Communiqué*, vol. 33, No. 1, pp 109-112, 1997.

[Spall98] J.C. Spall. Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization. *IEEE Transactions on Aerospace and Electronic Systems*, vol. 34, No.3 July 1998.

[Spall00] J.C. Spall. Stochastic Optimization, Stochastic Approximation and Simulated Annealing. Wiley Encyclopedia of Electrical and Electronics Engineering, A Wiley-Interscience Publication, vol. 20, pp 529-542, 2000.

[Stallings01] W. Stallings. Wireless Communications and Networks. Prentice-Hall, 2001.

[Stevens94] W.R. Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison Wesley, 1994.

[Stevens98] W.R. Stevens. Unix Network Programming, Networking APIs: Sockets and XTI, vol. 1, 2nd edition. Prentice Hall PTR, 1998.

[Tanenbaum02] A.S. Tanenbaum. Computer Networks, 4th edition. Prentice Hall PTR, 2002.

[TH98] J. Touch, S. Hotz. The X-Bone. *Proceeding of Third Global Internet Mini-Conference at Globecom*, Sydney, Australia, 1998.

[Tsunami] http://newsinfo.iu.edu/news/page/normal/588.html.

[VB00] A. Veres, M. Boda. The Chaotic Nature of TCP Congestion Control. *IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[Wasan69] M.T. Wasan. Stochastic Approximation. Cambridge University Press, 1969.

[Web100] http://www.web100.org.

[WebDarwin] I. Darwin. http://linux.oreillynet.com, Why Caldera Released Unix: A Brief History, March 2002.

[WS94] G.R. Wright, W.R. Stevens. TCP/IP Illustrated, Volume 2: The Implementation. Addison Wesley, 1995.

[YKZL00] Y.R. Yang, M.S. Kim, X. Zhang, S.S. Lam. Two Problems of TCP AIMD Congestion Control, 2000.

[YL00] Y.R. Yang, S.S. Lam. General AIMD Congestion Control. *Proceedings of International Conference on Network Protocols 2000*, November 2000.

APPENDIX SOURCE CODE FOR MQP() OF NETLETS IN CHAPTER 6

```
void CNetLet::ComputeRoutingTable(long int msgSize /*in bytes*/, int destNodeID)
{
	int *pResolvedIDs;
	int *pRestIDs;
	int resolved = 0;
	int i, j;
	double pathBW;
	CNode *pNode;
	FILE *fpRoutingTable;
	char filename[100] = "results/routingtable";
	double msgSizeinMb;
	int numPath = 0;
	int prevNodeID, currNodeID;
	int myID = m_MyNode.m_NodeID;
	msgSizeinMb = msgSize * 8.0 / 1000000.0; //convert to Mbits
	pResolvedIDs = new int[m_NumOfNodes];
	pRestIDs = new int[m_NumOfNodes];
	if((fpRoutingTable =  fopen(filename, "w")) == NULL)
	{
		printf("\nError create file results/routingtable!");
		exit(0);
	}
	fprintf(fpRoutingTable, "#Routing Table\n");
	fprintf(fpRoutingTable, "#SourceID DestID NextHopID ...\n\n\n");
	fprintf(fpRoutingTable, "#------------------------------------------------------------\n");
	printf("\n\n#Routing table\n");
	printf("#-----------------------------------------------------\n");
RepeatPathComputing:
	//The constituent paths of a multiple path for a given message size are computed by repeatedly
computing
	//the quickest path and removing it from the graph by reducing the appropriate bandwidths of the links.
	bool	breakout = false;
	InitRoutingTable();
	for(i = 0; i < m_NumOfNodes; i ++)
	{
		pRestIDs[i] = i;
		pResolvedIDs[i] = -1;
	}
	m_pNodeList[myID].m_dPathDelay = 0;
	m_pNodeList[myID].m_dPathBandwidth = INFINITY;
	pResolvedIDs[myID] = myID;
	pRestIDs[myID] = -1;
	resolved = 1;
	//Initialize node values from adjmatrix
	for(i = 0; i < m_NumOfNodes; i ++)
	{
		if(pRestIDs[i] == -1) continue;
		m_pNodeList[i].m_dPathDelay = m_pAdjacentMatrix[myID][i].m_LinkDelay;
		m_pNodeList[i].m_dPathBandwidth = m_pAdjacentMatrix[myID][i].m_Bandwidth;
		m_pNodeList[i].m_dPathLatency = m_pNodeList[i].m_dPathDelay + msgSizeinMb /
m_pNodeList[i].m_dPathBandwidth;
		m_pNodeList[i].m_pPredecessorNode = m_pNodeList + myID;
	}
	int indexMinKey = -1;
	double minKey = INFINITY;
	double newKey;
```

MQP() source code continues:

```
	while(resolved != m_NumOfNodes)
	{
		//find a node with the minimum key value in the rest set of nodes
		indexMinKey = -1;
		minKey = INFINITY;
		for(i = 0; i < m_NumOfNodes; i ++)
		{
			if(pRestIDs[i] == -1) continue;
			if(m_pNodeList[i].m_dPathLatency < minKey)
			{
				minKey = m_pNodeList[i].m_dPathLatency;
				indexMinKey = i;
			}
		}
		if(indexMinKey == -1)
		{
			printf("\nCan't obtain any more minimum paths for source node %d\n", myID);
			breakout = true;
			break;
		}
		//add this node with minimum key value to the resolved set of nodes
		pResolvedIDs[indexMinKey] = indexMinKey;
		pRestIDs[indexMinKey] = -1;
		resolved ++;
		//relax its neighbor nodes
		for(i = 0; i < m_NumOfNodes; i ++)
		{
			if(pRestIDs[i] == -1) continue;
			m_pNodeList[i].m_dPathLatency = m_pNodeList[i].m_dPathDelay +
msgSizeinMb / m_pNodeList[i].m_dPathBandwidth;
			pathBW = m_pNodeList[indexMinKey].m_dPathBandwidth <
m_pAdjacentMatrix[indexMinKey][i].m_Bandwidth ? m_pNodeList[indexMinKey].m_dPathBandwidth :
m_pAdjacentMatrix[indexMinKey][i].m_Bandwidth;
			newKey = msgSizeinMb / pathBW + m_pNodeList[indexMinKey].m_dPathDelay
+ m_pAdjacentMatrix[indexMinKey][i].m_LinkDelay;
			if(m_pNodeList[i].m_dPathLatency > newKey)
			{
				m_pNodeList[i].m_dPathDelay =
m_pNodeList[indexMinKey].m_dPathDelay + m_pAdjacentMatrix[indexMinKey][i].m_LinkDelay;
				m_pNodeList[i].m_dPathBandwidth = pathBW;
				m_pNodeList[i].m_dPathLatency = newKey;
				m_pNodeList[i].m_pPredecessorNode = m_pNodeList + indexMinKey;
			}
		}
	}//end of while
	//Build quickest paths
	if(breakout)
	goto Finish;
	for(i = 0; i < m_NumOfNodes; i ++)
	{
		if(i != destNodeID) continue;
		fprintf(fpRoutingTable, "#Minimum path from source ID: %d to destination ID: %d\n",
myID, i);
		fprintf(fpRoutingTable, "%d %d ", myID, i);
		printf("Minimum path from source ID: %d to destination ID: %d\n", myID, i);
		printf("%d %d: ", myID, i);
		m_RoutingTable.m_pPath[i].m_pSourceNode = m_pNodeList + myID;
		m_RoutingTable.m_pPath[i].m_pDestNode = m_pNodeList + i;
		m_RoutingTable.m_pPath[i].m_pNodeSeq = new int[m_NumOfNodes];
```

MQP() source code continues:

```
                        for(j = 0; j < m_NumOfNodes; j ++)
                                m_RoutingTable.m_pPath[i].m_pNodeSeq[j] = -1;
                        pNode = m_pNodeList[i].m_pPredecessorNode;
                        j = 0;
                        m_RoutingTable.m_pPath[i].m_pNodeSeq[j ++] = i;
                        while(pNode != NULL)
                        {
                                m_RoutingTable.m_pPath[i].m_pNodeSeq[j ++] = pNode->m_NodeID;
                                pNode = pNode->m_pPredecessorNode;
                        }
                        prevNodeID = myID;
                        for(j = m_NumOfNodes - 1; j >= 0; j --)
                        {
                                if(m_RoutingTable.m_pPath[i].m_pNodeSeq[j] != -1)
                                {
                                        if(m_RoutingTable.m_pPath[i].m_pNodeSeq[j] == i)
                                        {
                                                fprintf(fpRoutingTable, "%d\n",
m_RoutingTable.m_pPath[i].m_pNodeSeq[j]);
                                                fprintf(fpRoutingTable, "%lf %lf\n",
m_pNodeList[i].m_dPathDelay, m_pNodeList[i].m_dPathBandwidth);
                                                fprintf(fpRoutingTable, "#Total path latency: %lf = path delay:
%lf + path bandwidth constrained delay: (filesizeMb/pathBandwidth) %lf/%lf\n",
m_pNodeList[i].m_dPathLatency, m_pNodeList[i].m_dPathDelay, msgSizeinMb,
m_pNodeList[i].m_dPathBandwidth);
                                                printf("%d\n", m_RoutingTable.m_pPath[i].m_pNodeSeq[j]);
                                                printf("%lf %lf\n", m_pNodeList[i].m_dPathDelay,
m_pNodeList[i].m_dPathBandwidth);
                                                printf("#Total path latency: %lf = path delay: %lf + path
bandwidth constrained delay: (filesizeMb/pathBandwidth) %lf/%lf\n", m_pNodeList[i].m_dPathLatency,
m_pNodeList[i].m_dPathDelay, msgSizeinMb, m_pNodeList[i].m_dPathBandwidth);
                                        }
                                        else
                                        {
                                                if(m_RoutingTable.m_pPath[i].m_pNodeSeq[j] == myID)
                                                        continue;
                                                fprintf(fpRoutingTable, "%d ",
m_RoutingTable.m_pPath[i].m_pNodeSeq[j]);
                                                printf("%d --> ",
m_RoutingTable.m_pPath[i].m_pNodeSeq[j]);
                                        }
                                        //reduce the appropriate bandwidths of the links
                                        currNodeID = m_RoutingTable.m_pPath[i].m_pNodeSeq[j];
                                        m_pAdjacentMatrix[prevNodeID][currNodeID].m_Bandwidth =
m_pAdjacentMatrix[prevNodeID][currNodeID].m_Bandwidth - m_pNodeList[i].m_dPathBandwidth;
                                        prevNodeID = currNodeID;
                                }
                        }
                        printf("\n#-------------------------------------------------------\n");
                        fprintf(fpRoutingTable, "\n#-------------------------------------------------------\n");
                }       //end of for loop
                if(++ numPath < PATHNUMBER)
                        goto RepeatPathComputing;
Finish:
                fclose(fpRoutingTable);
                delete [] pResolvedIDs;
                delete [] pRestIDs;
}
```

VITA

Qishi Wu was born in Zhejiang, People's Republic of China, on December 22, 1972. He received his bachelor's degree in remote sensing and GIS from Zhejiang University, People's Republic of China, in 1995, and his master's degree in geomatics from Purdue University in 2000. He is currently a doctoral student in the Computer Science Department at Louisiana State University. He has been working in a joint network project between Louisiana State University and Oak Ridge National Laboratory. His research areas include computer networks, distributed sensor networks, algorithms, artificial intelligence, etc.